Authorization Architecture for SWoT

DRAGAN CABARKAPA

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

AALTO UNIVERSITY
School of Science and Technology
Faculty of Information and Natural Sciences
Department of Computer Science and Engineering

Dragan Cabarkapa

Authorization Architecture for SWoT

Master's Thesis
Espoo, August 26, 2013

DRAFT! — August 26, 2013 — DRAFT!

Supervisors: Professor Gerald Q. Maguire Jr., KTH Royal Institute of Technology, Sweden
Professor Tuomas Aura, Aalto University, Finland
Instructors: Christian Schaefer, Ericsson Research, Sweden
Farjola Zaloshnja, Ericsson Research, Sweden

Aalto University
School of Science

AALTO UNIVERSITY
School of Science and Technology
Faculty of Information and Natural Sciences
Department of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author:	Dragan Cabarkapa		
Title:	Authorization Architecture for SWoT		
Date:	August 26, 2013	**Pages:**	ix + 105
Professorship:	Data Communication Software	**Code:**	T-110
Supervisors:	Professor Gerald Q. Maguire Jr., Professor Tuomas Aura		
Instructors:	Christian Schaefer, Farjola Zalosnja		

Social Web of Things (SWoT) is a user centric framework which facilitates interaction between software agents deployed on smart things and in the cloud. Software agents deployed on smart things are remotely accessible, host sensitive resources, and often represent high value targets. SWoT currently does not feature adequate security mechanisms which could protect software agents from unauthorized access. In this thesis, we aim to rectify this deficiency by introducing platform independent, flexible, and user centric authorization mechanism in SWoT.

We derive requirements and design of abstract authorization architecture from the preceding seminal work performed in SENSEI project. SENSEI and SWoT share same problem domain, but while SENSEI addresses enterprise use cases SWoT focusses on consumer use cases. This single but fundamental difference motivates adaptations of SENSEI contributions for application in SWoT. To realize concrete authorization architecture we perform extensive study of various authorization solutions. Results of our study indicate that novel User Managed Access (UMA) protocol represents promising solution for SWoT.

We present the Authorization as a Service solution for SWoT framework, based on UMA protocol. This solution enables users to manage and control communication between software agents deployed on smart things and in the cloud from single centralized location. It also features runtime association of software agents, management, evaluation, and enforcement of access permissions for resources provided by software agents.

Keywords:	SWoT, Security, Authorization, User Managed Access
Language:	English

Aalto-universitetet
Högskolan för
teknikvetenskaper

AALTO-UNIVERSITETET
Tekniska Högskolan
Fakulteten for Informations och Naturvetenskaper
Utbildningsprogrammet för Datateknik

SAMMANFATTNING AV
DIPLOMARBETE

Utfört av:	Dragan Cabarkapa		
Arbetets namn:	!Fixme **Thesis title in Swedish.** Fixme!		
Datum:	2013	**Sidantal:**	ix + 105
Professur:	Datakommunikationsprogram	**Kod:**	T-110
Övervakare:	Professor Gerald Q. Maguire Jr., Professor Tuomas Aura		
Handledare:	Christian Schaefer, Farjola Zalosnja		
!Fixme **Write thesis abstract in Swedish.** Fixme!			
Nyckelord:	!Fixme **Keywords in Swedish.** Fixme!		
Språk:	Engelska		

Acknowledgements

First of all, I would like to thank Ericsson AB for providing me an opportunity to work on this exceptional research project. My gratitude goes to my supervisors, Christian Schaefer and Farjola Zalosnja for their continual support. Without their advices through countless discussions this thesis would not be possible.

I would like to specially thank my supervisors, Prof. Gerald Q. Maguire Jr. and Prof. Tuomas Aura for their excellent supervision and valuable feedback throughout this thesis work.

I owe gratitude to NordSecMob program coordinators first for creating it and second for providing me an opportunity to be part of it. I am also grateful for their patience and consideration when I had difficulties keeping with the initial thesis schedule.

My biggest thanks goes to my family, my girlfriend, and my friends whose support helped me navigate through educational endeavors, gave me strength to overcome all obstacles, and achieve this final accomplishment.

Stockholm, August 26, 2013

Dragan Cabarkapa

Abbreviations and Acronyms

AAA	Authentication, Authorization, Accounting
AAPI	Authorization API
AAT	Authorization Access Token
ABAC	Attribute Based Access Control
ALS	Application Layer Service
API	Application Programming Interface
AS	Authorization Server
CL	Capability List
COS	Cloud Operating System
DB	Database
DS	Directory Service
DoS	Denial of Service
FP	Framework Provider
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
LCA	Light Controller Application
LDA	Light Device Application
OASIS	Organization for the Advancement of Structured Information Standards
OS	Operating System
PAP	Policy Administration Point
PAPI	Protection API
PAT	Protection Access Token
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point

QoS	Quality of Service
RBAC	Role Based Access Control
REST	Representational State Transfer
RLS	Resource Layer Service
RP	Resource Provider
RPT	Requesting Party Token
RS	Resource Server
RU	Resource User
SAC	Social Access Controller
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query Language
SSO	Single Sign On
SSU	Spring Security UMA
SWoT	Social Web of Things
TCB	Trusted Computing Base
TLS	Transport Layer Security
UMA	User Managed Access
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
WoT	Web of Things
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

Contents

Abbreviations and Acronyms **v**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem statement 2
1.3 Research Goals and Contributions 3
1.4 Thesis scope and approach 4
1.5 Thesis structure 5

2 Background **6**
2.1 SWoT system 7
2.1.1 Architecture Model 7
2.1.2 Actors and Artefacts 7
2.1.3 Interaction Model 12
2.1.4 Trust Model 13
2.1.5 Threat Model 14
2.2 Related work 15

3 Study of security mechanisms **21**
3.1 Distributed systems and security architecture 21
3.2 Security components and procedures 23
3.2.1 Identity management system 23
3.2.1.1 Registration 25
3.2.1.2 Authentication 26
3.2.2 Access management system 26
3.2.2.1 Authorization 27
3.2.2.2 Access control 29
3.2.2.3 Delegation 29
3.3 Security mechanisms 30
3.3.1 Kerberos 32
3.3.2 Security Assertion Markup Language 33

3.3.2.1 SAML analysis . . . 34
3.3.3 XACML . . . 34
3.3.3.1 XACML analysis . . . 35
3.3.4 OAuth . . . 36
3.3.4.1 OAuth analysis . . . 38
3.3.5 User Managed Access Profile of OAuth 2.0 . . . 39
3.3.5.1 UMA analysis . . . 43
3.4 Security mechanisms analysis . . . 43

4 System specification **47**
4.1 Bootstrap phase . . . 47
4.1.1 Service registration . . . 47
4.1.2 Service association . . . 50
4.2 Configuration phase . . . 52
4.2.1 Resource registration . . . 52
4.2.2 Resource discovery . . . 53
4.2.3 Resource administration . . . 55
4.3 Operation phase . . . 56
4.3.1 Authentication . . . 56
4.3.2 Authorization . . . 58
4.3.3 Authorization evaluation . . . 60

5 Implementation **63**
5.1 Libraries . . . 63
5.2 UMA Resource Server . . . 65
5.3 UMA Client . . . 67
5.4 UMA Authorization Server . . . 68

6 Analysis and Discussion **73**
6.1 Functional analysis . . . 73
6.1.1 Use case components . . . 73
6.1.2 Evaluation test cases . . . 75
6.2 Security analysis . . . 77
6.2.1 Overview of assumptions and features . . . 77
6.2.2 UMA threats . . . 78
6.3 Performance analysis . . . 82
6.3.1 UMA flows . . . 82
6.3.1.1 Obtaining RPT flow . . . 82
6.3.1.2 First resource access flow . . . 83
6.3.1.3 Second and subsequent resource access flow . 84
6.3.2 Performance of UMA flows . . . 85

6.3.3 Parameters affecting performance 87
6.3.3.1 Permission expiration period 89
6.3.3.2 Resource granularity 89
6.3.3.3 Application interaction patterns 90
6.4 Discussion 91

7 Conclusions **93**
7.1 Summary 93
7.2 Future Work 94
7.3 Reflections 95

If we had a reliable way to label our toys good and bad, it would be easy to regulate technology wisely. But we can rarely see far enough ahead to know which road leads to damnation. Whoever concerns himself with big technology, either to push it forward or to stop it, is gambling in human lives.

Freeman Dyson, Disturbing the Universe

As the planet's population continues to increase, it becomes even more important for people to become stewards of the earth and its resources. In addition, people desire to live healthy, fulfilling, and comfortable lives for themselves, their families, and those they care about. By combining the ability of the next evolution of the Internet (IoT) to sense, collect, transmit, analyze, and distribute data on a massive scale with the way people process information, humanity will have the knowledge and wisdom it needs not only to survive, but to thrive in the coming months, years, decades, and centuries.

Dave Evans, The Internet of Things

There was of course no way of knowing whether you were being watched at any given moment. How often, or on what system, the Thought Police plugged in on any individual wire was guesswork. It was even conceivable that they watched everybody all the time. But at any rate they could plug in your wire whenever they wanted to. You had to live-did live, from habit that became instinct-in the assumption that every sound you made was overheard, and, except in darkness, every movement scrutinized.

George Orwell, 1984

It is only by the rational use of technology; to control and guide what technology is doing; that we can keep any hopes of a social life more desirable than our own: or in fact of a social life which is not appalling to imagine.

Charles Percy Snow

Chapter 1

Introduction

Social Web of Things (SWoT) [1] is a research project lead by Ericsson. The primary goal of this project is to explore opportunities and challenges of applying Internet of Things (IoT) concepts in smart home application domain. The research within the project is performed through realization of the SWoT framework, a consumer oriented middleware solution. The primary purpose of the SWoT framework is to enable integration of cloud and smart device hosted services which are part of different administrative domains. Within SWoT project researchers are exploring and solving various challenges. For example, they are developing novel interface paradigms to facilitate user understanding and monitoring of service to service interactions, they are developing mechanisms for autonomous integration of heterogeneous services, and they are evaluating different interaction paradigms. However, their focus has been primarily on functional aspects of the SWoT platform. Due to this some important non-functional aspects, such as security, have not been properly considered.

1.1 Motivation

It is well known that Internet based solutions, both consumer or enterprise oriented, should not be deployed in production without suitable security mechanisms which will protect them from malicious adversaries. Otherwise actions of malicious adversaries may result in system compromise and damages in extent proportional to system capabilities. IoT solutions are not an exception to this rule. On the contrary, security aspect in realization of IoT solutions is even more emphasized due to their inherent characteristics (they are able to actuate, sense, and record information related to the physical environment, they are remotely controllable, and they

are pervasive).

Security issues for smart devices which are remotely controllable and globally accessible have for some time been identified and emphasized in different research works [2][3][4]. Recent security incidents related to the early stage IoT solutions, prove that identified security risks are real and still mostly unaddressed [5][6][7]. Despite this, researchers in IoT domain mainly focus on challenges related to the functional requirements in their solutions and decide to neglect present security requirements [8][9].

IoT products feature radically different characteristics from currently typical Web solutions (such as Web services): they have constrained physical resources, they are expected to be managed by end users, which are not experts, and their number continuously increases [10][11]. Moreover, in contrast to Web solutions where one to one interactions are typical, in IoT solutions many to many interactions are more emphasized as implementation of particular use case scenario often includes interactions with multiple smart devices. Current well established Web based security mechanisms have not been designed to cope with these characteristics and use cases. Therefore, question which researchers often ask is: are current web based security mechanisms capable enough to individually or in combination satisfy security requirements present in IoT solutions.

In the SWoT project the primary focus in the development of SWoT framework has been on functional requirements. Therefore, all previously mentioned security issues, requirements, and questions are also largely unaddressed. This thesis project represents an initial effort in analysing and addressing subset of SWoT's security requirements.

1.2 Problem statement

The SWoT framework is envisaged to enable service to service interactions across different administrative domains. Overall system security risks are high due to inherent capabilities of remotely controllable smart devices. Therefore, adequate security mechanisms need to be in place. Security solution to be developed has to take into account following high level requirements:

- The system which SWoT framework defines is a distributed system that consists of numerous services deployed in the cloud and on the smart devices. The common trait of all these services is that they act on behalf of specific user. The interaction between services is governed by the user with the help of security solution provided by the SWoT

framework. Therefore, while the overall system is distributed, system administration functionality is centralized.

- Both SWoT framework and security solution to be developed as its integral part are expected to be used by the non expert users. Since users may not be familiar with inner workings of security mechanisms it is important that the system from user perspective is characterized as simple, unobtrusive, and transparent. Therefore, preferred security mechanisms are those which reduce user effort in system administration and facilitate more autonomous establishment of security context that spans over and includes different SWoT system entities.

- The SWoT framework relies on Web based technologies for service integration. Therefore, developed security solution should rely on same set of technologies in order to enable frictionless third party application development and integration.

The combination of characteristics and requirements present in the SWoT system, which is a *distributed*, through inclusion of smart devices *ubiquitous*, and by the enactment of the user to be main administration authority *user centric*, represents serious challenge in the realization of an adequate security solution.

1.3 Research Goals and Contributions

This thesis project studies the challenges in realization of the authorization architecture suitable for SWoT framework. Our study has three main goals. The first goal is to design authorization architecture for the SWoT framework. The second goal of this thesis project is to study and identify suitable security protocols for previously defined authorization architecture. The third goal of this thesis project is to implement and evaluate a proof of concept solution. Evaluation study should determine the impact of proposed solution on individual system entities and overall system from usability, security, and performance perspective.

The first contribution of this thesis project is the design of the authorization architecture for the SWoT framework. Our solution is largely based on the authorization architecture design developed in SENSEI project [12]. However, we have also adapted architecture suggested in SENSEI as well as related artifacts, such as authorization system requirements, trust model, and threat model, to take into account characteristics of the SWoT framework.

The second contribution of this thesis project is the extensive study of security mechanisms including Kerberos, SAML, XACML, OAuth, and UMA. In this study we have evaluated security mechanisms based on several different criteria including: their support for user centric authorization, their support for runtime introduction of service providers, and their usage of web based technologies.

The third contribution of this thesis project is our implementation of UMA framework. Our implementation is based on Spring Security and Spring Security OAuth libraries. It provides modules which facilitate implementation of authorization server, client, and resource server entities. In addition, we provide results of functional, security, and performance evaluation of our solution.

1.4 Thesis scope and approach

For the design and development of the fully functional SWoT security architecture it would be necessary to consider numerous security aspects (such as authentication, authorisation, and accounting) in different computing domains (embedded computing and cloud computing) and security domains (physical, network, and computer security). Since this is an immensely complex task and inappropriate for a master's thesis project, our approach here is much more modest and narrower in scope.

In this thesis project we focus on realization of the authorization component which is considered core component of the security architecture. Even in this reduced scope we do not consider all problem aspects. For example, we do not discuss how authorization attributes are encoded and exchanged. We also do not consider how access control policies should be defined and evaluated. We only partially consider registration (trust establishment) and authentication system components in extent required to elicit authorization system inputs.

We do not consider issues related to the physical security nor we take into account resource limitations of smart devices. We assume presence of network security mechanisms which provide confidentiality and integrity of system entities' communication. Readers interested in these closely related security aspects are encouraged to read the deliverables from the SENSEI project, which discuss these and many other security aspects extensively in [12].

The SWoT framework development at the time this thesis project began had already progressed significantly so that even small changes in the implemented security architecture or mechanisms would have a ripple effect

on whole framework. As a result, the SWoT management decided to run this project independently from the main SWoT project with the goal of integrating results later if evaluation of newly developed security architecture provided promising results.

This clean slate approach resulted in several concrete benefits. First we were able to reduce the scope of the work and make it more appropriate to the thesis project. Second we were able to reduce the complexity of our prototype system's design, implementation, and evaluation. Finally, since the problem that we are addressing is quite common and our analysis and solution decoupled from the specifics of the current SWoT implementation, our contributions are applicable in a wider range of domains.

1.5 Thesis structure

The rest of the thesis is structured as follows. In Chapter 2 we describe SWoT system architecture, actors, artefacts, interaction, trust, and threat models. In Chapter 3 we survey security protocols with the goal of identifying suitable solutions for realization of the proposed SWoT authorization architecture. In Chapter 4 we describe how we integrate and apply selected security mechanisms. Chapter 5 documents our implementation efforts with a particular focus on implementation of the UMA framework. In Chapter 6 we present functional, security, and performance evaluation of our system. Chapter 7 provides a summary of thesis contributions and suggests directions for future work.

Chapter 2

Background

Internet of Things (IoT) is a vision of the Internet with the smart devices integrated into the common communication infrastructure [13][14][15]. In its narrow definition, IoT research domain covers technical solutions which enable integration of smart devices on the infrastructure level. In its more broader definition IoT research domain also covers any application level solution which incorporates or relies on the capabilities of smart devices. This broad definition includes more specific research domains, such as Web of Things [16][17][18], in which researchers adopt Web based technologies for handling application level integration.

The concept of integration of physical and digital worlds is not new. Researchers in domains of ubiquitous computing, pervasive computing, and ambient intelligence, have explored for some time now challenges and solutions similar to those present in IoT. However, maturing hardware technology, wide adoption of mobile devices, coupled with financial support from numerous small and large business entities is bringing us closer to the realization of IoT.

Both the potential and challenges for realization of IoT solutions lie in characteristics of smart devices [13]. Smart devices feature specific functions and capabilities. Each individual device is considered important, even though it features same functionality as some other smart device. They are deployed in physical environment, and manufactured, owned, and operated by different entities. With the exception of the most simple use case scenarios, the systems developed to realize IoT applications are highly distributed systems. This is mainly due to moving application logic from the smart devices to the higher level services [8], and more prominent use of mashup like applications for complex automation scenarios.

2.1 SWoT system

Social Web of Things (SWoT) is an Ericsson research project in the domain of ambient intelligence. The main deliverable is a platform which enables connecting services deployed on things and in the cloud. In contrast to industry oriented platforms such as ThingWorx [19] and Axeda [20], SWoT is focused on the consumer market in which the end users are individuals. This perspective introduces unique challenges and drives the need for equally unique solutions in various domains including security and authorization.

2.1.1 Architecture Model

Within the context of SWoT project two alternative architecture models are recognized: loosely coupled and tightly coupled architecture model (see Figure 2.1). Both of the architecture models share same control plane which includes security and service discovery related interactions. However, their data planes which include interactions directly related to functions of smart devices being part of the SWoT domain, differ. In loosely coupled architecture data plane interactions are performed directly between services part of the SWoT domain, while in tightly coupled architecture data plane interactions are mediated by the SWoT framework services. This mediation in tightly coupled architecture results in three different data plane parts, marked in Figure 2.1 as data plane 2a, 2b, and 2c. Each of these data plane's parts viewed individually has similar configuration of actors and functional interactions as in loosely coupled architecture. Due to this we consider tightly coupled architecture model to be more complex version of loosely coupled architecture model and that the prerequisite for the realization of tightly coupled architecture is the realization of loosely coupled architecture. This thesis project explores realization of the control plane's security service based on the requirements present in loosely coupled architecture model. The realization of the security service that satisfies more complex requirements present in tightly coupled architecture model is part of future work.

2.1.2 Actors and Artefacts

A resource layer service (RLS) is a term used in SWoT project to refer to a class of services which are provided on smart devices. RLSs expose resources through which other SWoT entities take advantage of sensing and/or actuating capabilities built into smart devices hosting RLSs. The functionality offered through RLSs is closely associated with the function of the smart device. For example, smart lock device [21] may host RLS

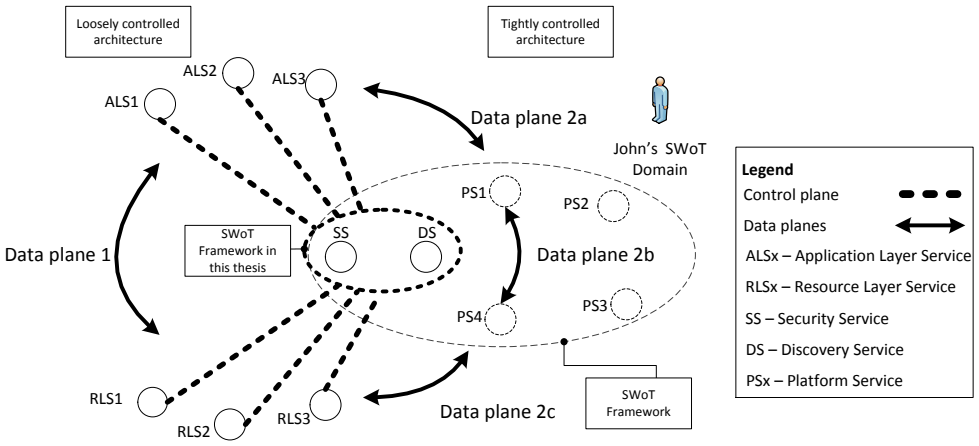


Figure 2.1: Left: Loosely controlled architecture with distributed service delivery (1). Right: Tightly controlled architecture with centralized service delivery (2a and 2b).

which offers functionality for locking and unlocking operation, smart light device [22] may provide RLS which offers functionality for turning on or turning off the light. The main characteristic of RLS is that they do not host complex application logic. The entities hosting and providing complex application logic are RLS clients. Moving application logic to the client side decouples infrastructure and application layer. Moreover, it provides benefits of faster and friction less application development as well as easier reuse of smart devices by multiple applications [8]. However, this approach has many implications as it diverges from conventional approach applied in Web, in which application logic is located on the server. The main implication is that users are expected to take advantage of smart device capabilities through RLS clients rather than in direct interaction with the RLSs. Consequently, the application programming interface (API) designed for service to service communication becomes the primary interface of RSLs. This is in contrast to current web services where primary interfaces are primarily consumed by browsers on behalf of human users.

For the purpose of this thesis project RLSs are considered to be implemented as Representational State Transfer (REST) services. In this aspect they are similar to Web services which offer REST APIs. This particular implementation approach is underlines research efforts in Web of Things (WoT) research community [16][17][18][8].

An application layer service (ALS) is a term used in SWoT project

to refer to a class of services which are RLS clients and have following characteristics: they are multiuser web applications, they are managed by ALS developer, and they are deployed as cloud services. ALSs are developed to interact with particular types of RLSs. Therefore, their functionality is closely related to the capabilities of RLSs with which they interact. However, ALS enhance functionality of RLSs by implementing more complex application logic. In addition, they can simultaneously interact with multiple RLSs hosted on different smart devices. These ALSs capabilities enable realization of more complex use case scenarios. For example, particular ALS may enable users to control multiple smart lights with single action, or even automate this action to be invoked at specific time. Other ALS may enable users to monitor energy consumption of their smart appliances in raw or processed form, and based on this information to compete with other users in minimizing their energy consumption. ALSs can be classified as mashup service. However, they are more complex from mashup services typical today as they interact with larger number of services. For example, today typical mashup service combines user information from three or more social network services which are same for all users, while an ALS combines information from tens of different smart devices, which are of the same RLS type, but still distinct RLS instances.

SWoT framework is an integration service enabling interaction between RLSs and ALSs. Moreover, it is a single user web application. It is expected that each user will have their own SWoT framework instance. These instances are physically isolated from other SWoT framework instances to preclude SWoT framework resource misuse and reduce potential damage in case of SWoT framework instance compromise. SWoT framework user centric design is amenable for different deployment scenarios (cloud hosted, or self hosted) and different business models (subscription, and profit sharing).

SWoT framework in scope considered here (see Figure 2.1) features two components: discovery service and security service. Discovery service function is to facilitate configuration and operation of ALSs. ALSs in interaction with the discovery service may obtain information related to RLSs, which are of interest to ALSs, and which belong to particular user. Presence of discovery service obviates the need for the user to manually inform ALSs about the location, version, and type of RLSs that user manages. For example, instead of user entering information about each smart light service (set of RLSs) that he or she manages at the smart light controller service (single ALS), smart light controller can retrieve this information automatically in interaction with the discovery service. For discovery service to be able to provide RLS related information it first needs to obtain it. This is achieved through interaction with RLSs which provide this information in

specified format and by following predetermined protocol.

Security service, which realization is the main goal of this thesis project, is envisaged as core component of the SWoT framework. Its purpose is to protect resources of entities belonging to SWoT administrative domain. This includes resources which are known at design time, such as functionality for RLS resources registration and discovery provided by discovery service, and resources which are added to the administrative domain at run time, such as resources provided by RLSs. In fulfilling this purpose security service needs to provide four different functions. First function of security service is to enable user driven establishment of authorization context which would span over different RLSs and ALSs and across different administrative domains. This effectively enables the user to add/remove new RLS or ALS to/from his/her SWoT domain at any time. Second function of security service is to provide means using which user may define access control policies. Closely related to this function is management of different attributes related to entity accessing the resource, resource being accessed, or even environment conditions, such as time of day, as access control policies may be based on these attributes. Third function of security service is to evaluate defined access control policies and derive adequate access decisions upon adequate request. Fourth function of security service is to enforce previously derived access control decisions in cooperation with SWoT domain entities hosting the resources which are to be protected.

For actors in context of particular domain we use terms resource provider (RP), resource user (RU), and framework provider (FP) to refer to RLS, ALS, and SWoT framework instances acting on behalf of particular person respectively (see Figure 2.2). This terminology enables us to refer to particular person - software agent combination. For example, specific smart light controller (ALS) may be integral part of multiple RUs as it can act on behalf of various users while particular RU identifies both the ALS and the user on which behalf this ALS acts. Even in the case of single user systems such as SWoT framework, it is helpful to express that particular instance is acting on behalf of specific person as this is the only distinction between different SWoT framework instances. This terminology helps us also to define the meaning of SWoT domain, which in this thesis project is viewed as a logical unit consisting of a set of RPs, a set of RUs, and a single FP which all act on behalf of the same person. Furthermore, this person interacts with software entities in different roles: it is an administrator in RPs, a user in RUs, and a user in FP.

Main artifacts in SWoT system are a resource, a resource description, and a service description.

A resource, abstractly defined, is an representation of a functionality

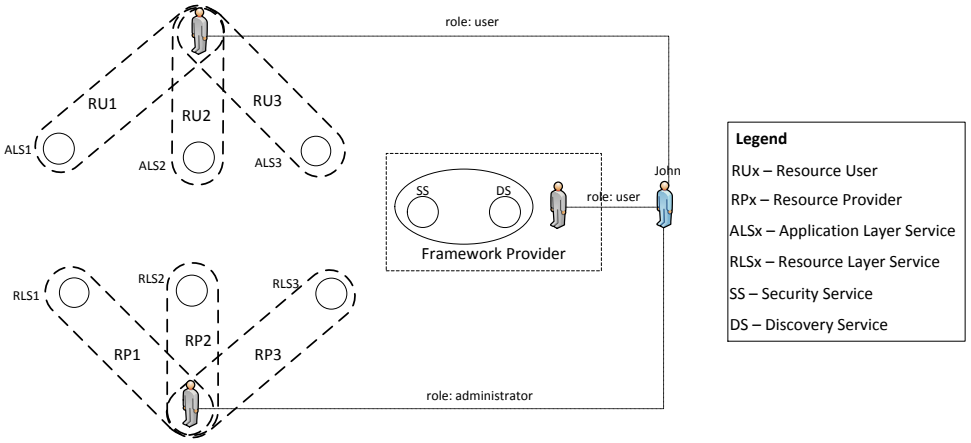


Figure 2.2: Actors in SWoT domain.

which provides information or performs particular action based on specific input. Resources in SWoT represent different capabilities of smart devices, such as sensing or actuation capabilities. Other services through interaction with the resources gather information from or influence processes in the physical environment. In addition to usage of resource concept to abstract smart device's physical capabilities, the resource concept is also used to abstract smart device management functionality. This enables the use of the same mechanisms for interacting with the resources as well as for managing smart devices and resources that they host. Interactions with resources are performed through a uniform REST interface and communicated using Hypertext Transfer Protocol (HTTP). Each resource is identified by a globally unique identifier in form of a Uniform Resource Identifier (URI). In addition each resource is associated with the corresponding resource description.

A resource description is a SWoT artefact which enables the SWoT framework to manage and advertise resources part of SWoT domain. Resource description contains metadata relevant for a particular resource such as resource identifier, a resource type, and other semantic information describing resource's capabilities. In addition, the resource description features information relevant for access control mechanisms, such as operations and permissions required for interaction with the particular resource. SWoT framework obtains resource descriptions from RLS when a new RLS is added to a SWoT domain or new RLS resource is created.

A service description is a SWoT artefact which enables the framework

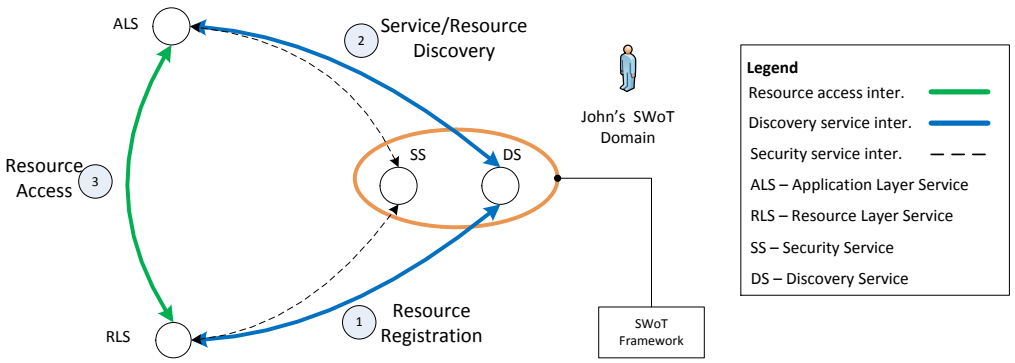


Figure 2.3: Interaction model.

to manage and advertise RLSs and ALSs part of the SWoT domain. Service description contains metadata, such as service identifier, type, version, user friendly description, icon, and etc. Service descriptions are used for facilitating interaction between RLS and ALS and for populating SWoT framework service management user interface.

2.1.3 Interaction Model

To facilitate interaction between ALS and RLS, SWoT framework provides discovery service functionality. This functionality enables autonomous discovery of RLSs by ALSs and obviates the need for human user involvement. However, for discovery service to advertise RLSs and resources which they offer, RLSs need first to provide required information in predetermined format. The interaction in which RLSs provide resource descriptions is known as resource registration (step 1 in Figure 2.3). ALSs interested in particular type of RLS or resource provided by the RLSs may initiate service/resource discovery process (step 2 in Figure 2.3). In this process ALSs interact with the SWoT framework's DS and retrieve service or resource description artefacts containing various information (such as location of resource end points, RLS version, or RLS type). ALSs with this information in possession, may initiate interaction with the corresponding RLS. The interaction in which ALS directly communicates with the RLS and accesses particular resource (step 3 in Figure 2.3) represents core system interaction as at this step smart device's actuation and sensing capabilities are actually utilized. All other interactions performed within the system are necessary but serve as system support functionality.

Previously described interactions are time separated and do not happen

with same frequency. For example, resource registration interaction is performed only when resource provided by a RLS is modified in some way (created, removed, or its description is changed due to smart device firmware update). Similarly service/resource discovery interaction is performed during ALS initial configuration and later on regular time intervals or in reaction to particular human user actions. In contrast to these interactions, resource access interaction, is expected to happen much more often.

2.1.4 Trust Model

When the smart device and application level services which utilize capabilities of the device are managed by single entity security trust model is relatively simple. However, when smart device and application level services which interact with the device are managed by different entities, security trust model becomes more complex and consequently complexity of the security solution required to address it increases. Since SWoT needs to integrate services from different administrative domains, its security architecture needs to satisfy more complex set of security requirements.

Every system has trusted components on which it relies. In SWoT system these trusted components are SWoT framework which is trusted for the overall system operation, and RLSs which are trusted for system operations that rely on RLSs. System cannot avoid trusting RLSs as they represent original source of information and final executor of actions. In some respect, this is similar to operating systems which trust and rely on physical hardware devices (such as hard disks, network controllers) to operate as expected.

SWoT system also features components which are initially totally untrusted. These are ALSs which through interaction with the SWoT framework and under user defined conditions obtain specific permissions and trust in extent defined by the granted permissions.

Based on two different trust models (see Section [12]) two alternative approaches for handling authorization on RP's side are envisaged. These are: identity based approach, where access privileges are granted and associated with the identity of the RU; and billing based approach, where access is allowed to any RU willing to pay. Based on these two approaches, identity based and billing based trust model are defined respectively.

Both models are applicable in studied problem domain. However, the identity based trust model is more generic [23]. Taking into account this fact and due to our limited resources we have made the decision to focus our study here only on identity based trust model. Study of requirements, architecture and mechanisms required to realize solution based on billing trust model is left for future work. However, we believe that results provided in this thesis

will facilitate this future endeavor as two trust models have much in common.

2.1.5 Threat Model

We provide here a high level threat model based on previously described interaction model. Interactions present in our interaction model represent subset of those found in SENSEI interaction model. As a result threats listed in Table 2.1 represent subset of those identified and documented in Section 8.5.2 of [23]. Each row contains identified threat, attackers benefit, impact and likelihood of the threat. Impact and likelihood are quantitatively described with *high*, *medium* and *low* grades. In addition each row contains short list of main mitigation approaches with their classification: *Te* - technical solution, *Le* - legal solution, *Tr* - part of trusted computing base. To indicate how threats in our model map to those present in SENSEI threat model we provide in index column in parentheses references to indexes of corresponding threats in SENSEI threat model. We here describe threats listed in Table 2.1 grouped by their mitigation approaches.

Technical measures - SWoT entities are part of logical domain and they are not physically isolated from the rest of the Internet. Which means that FP or RPs can be remotely accessed by both authorized and unauthorized entities. To mitigate threats in which third party or SWoT entity impersonates other SWoT entity (threats 1, 2, 3) authentication is used. With the help of authentication entities providing resources FP and RPs know identity of RUs. However knowing identity is only the prerequisite. To mitigate threats in which entity, with asserted identity, accesses resources that it is not supposed to access (threats 4, 5) authorization is used. With the help of authorization FP and RPs know which RUs can access which resources and under which conditions. Authentication and authorization combined provide basic security architecture. However this security architecture is not considered complete without accounting mechanisms which enable tracing all security interesting events in the system. Without it detecting flaws in configurations of authentication and authorization mechanisms (threats 4, 5) or attack attempts whether successful or unsuccessful (threats 6, 7) would be impossible.

Denial of Service (DoS) measures - Denial of service (DoS) is a threat directed against service availability (threats 6, 7). Attackers can cause DoS by applying particular form of the DoS attacks such as resource exhaustion attack [24]. Both FP and RPs can be contacted directly by malicious entities and therefore they may be susceptible to this threat. We expect that FP will have at its disposal significantly more resources compared to RPs individually and therefore will be more capable to sustain attacks of this type. However

even resources at the disposal of FP will be limited since SWoT is envisaged as user centered system. Impact of this threat on the SWoT system depends whether RP or FP is attacked. In case of successful attack on individual RP impact on the system depends on importance of particular RP. In case of successful attack on FP whole system stops functioning. Addressing this threat is challenging since both its impact and likelihood are high while effectiveness of defence mechanisms is limited.

Non-technical measures - Even though technical measures are irreplaceable part for creating secure systems they alone are insufficient. This is especially true for a system such as SWoT in which entities produced by different manufacturers are not only involved in its operation but they are trusted part of the system. For example in SWoT system RPs are trusted to comply with prescribed security protocol. To mitigate this trust being broken it is necessary to introduce combination of technical and legal measures e.g. certification and quality assurance program for RPs software agents or legal contracts describing who should be held accountable in situations falling outside system's normal operation. Many of the threats listed in Table 2.1 require this type of measures (threats 4,5,6,7,8,9,10).

Technical measures for securing the system may be complex. However when considered in combination with legal measures complexity increases significantly. Research works focusing on the intersection of these two domains are rare but available. Contributions from researchers range from those low level and directly associated with technical measures [25] to those with primary focus on legal issues [26] [4] [27].

2.2 Related work

SENSEI [**28**] was a large scale project which explored challenges and required solutions to enable integration of physical and digital worlds. The main project deliverable was SENSEI framework, a middleware solution designed to broker interaction between services deployed on physical devices and those services deployed in the cloud. SENSEI and SWoT projects share similar goals, but their scopes and adopted design approaches differ. For example, SENSEI project scope covers federation of different SENSEI framework instances while currently this use case scenario is not envisaged for SWoT framework. Moreover, SENSEI framework has been designed to be organization centric solution while SWoT framework is user centric solution. These fundamental differences are reflected in projects' deliverables. While SENSEI solutions for SWoT use case are overly complex and not directly applicable, the SENSEI project deliverables discussing system and security

architecture design [23][12] provide many useful contributions. Some of these contributions have been incorporated into the SWoT models (architecture, interaction, trust, and threat model).

Cloud operating system (COS) [29] is a vision of novel class of user centric cloud hosted solutions. COS is introduced by analogy with the classical operating systems (such as Linux, Windows, and OS X) which execute in a isolated computation space defined by personal computer. COS is also envisaged to execute in a isolated user specific computation space. However, this user space is not tied to the specific physical hardware. There are many properties in which COS differs from classical operating systems. For example, COS is always on, it is easily expandable with additional storage and computation resources, and it is always available independent of users physical location. However, the characteristics which define COS include its capability to: establish and manage identities of distributed entities in the context of specific COS instance, mediate and abstract different Web APIs, and manage COS owner's personal data.

SWoT framework as a user centric solution is similar to COS. The security solution which we have developed in this thesis project is based on UMA security protocol, the same mechanism used in COS to establish authorization context spanning over different entities and across different administrative domains. However, UMA is not complete solution as it covers only system external interactions. In our solution we also discuss system internal security mechanisms, such as ABAC access control model based on XACML and its integration with UMA protocol. Moreover, we provide more detail depiction of these security mechanisms characteristics as well as their application in SWoT specific use case.

Solutions and research works from ubiquitous systems domain, such as Gaia OS [30][31][32], provide extensive study of requirements, security models, and security mechanisms used to realize concepts of smart and active spaces. The central concept in Gaia OS is that of physical space. This physical space has its administrator who defines policies governing interactions between users, devices, and software entities operating within this physical space. Gaia OS security solution takes into account security requirements from both physical and virtual environment. In addition, Al-Muhtadi et al. [31] describe and realize context aware access control system for Gaia OS with all required supporting infrastructure and software services. Gaia OS and its security component represent perfect example of how complex security requirements and corresponding security solutions are in systems which integrate virtual (software) entities with physical devices and their capabilities.

SWoT framework, compared to Gaia OS, is still in its infancy stage.

Moreover, it features different design approach as the user (device owner) and not physical space is the integration point or central concept. Within SWoT project, at current stage, physical requirements both in system and security architecture development are not considered, even though their importance is recognized. Our security solution developed in this thesis project integrates state of the art security mechanism (OAuth, UMA, and XACML), which have not been available at the time when Gaia OS was developed. In addition, our solution relies on contemporary communication and application technologies (Internet, Web) that were available when Gaia OS was developed but were at that time considered inappropriate for application in Gaia OS use case scenarios. We view our security solution as a basic infrastructure for future Gaia OS like system enhanced and developed with more modern technologies.

Social Access Controller (SAC) [**33**] is a user centric solution which enables users to share with other users access to their devices. It relies on functionality provided by popular social networking sites for user authentication. In addition, social networks are used for advertising and discovery of devices. SAC was proposed to address limitations of centralized platforms which act as a centralized data repositories and restrict direct interaction with devices. SAC addresses this limitation by transparently exposing device resources and proxying all the communication to/from the device. To be able to access devices, SAC requires from the owner to manually configure SAC with credentials necessary to access each device. Moreover, SAC relies on crawling resource endpoints exposed on the device to discover resource which need to be protected. Based on the information retrieved in this process user specifies access controls rules which SAC enforces.

Guinard et al. provide very limited description of SAC security functionality. Moreover, they rely on unconventional approach for handling resource discovery for the purpose of retrieving input information for access control system. Our security solution is comprehensive, and based on current established or draft standards. It also addresses more complex use cases which is sharing access permissions with third party software agents instead of users relying on browsers. In addition, our solution is amenable for implementing fine grained and context aware access control.

Smart home gateway architecture [**34**] integrates heterogeneous protocols used in home networks. This gateway architecture is designed to proxy the communication between cloud services and physical devices deployed within the home. In their solution Kim et al. integrate a semantic model of a smart home which enables integration of new applications at runtime. Moreover, they introduce novel access control mechanism based on XACML. Policy model which is integral part of this access control mechanism

includes roles, permissions, and attributes (such as device type, device status, time, and location). Policy evaluation is supported by use of ontologies.

The SWoT framework and solution of Kim et al. share same problem domain. However, approaches used in these projects for addressing problem domain challenges differ significantly. For example, while smart home gateway is a solution that integrates web technologies on one side with current smart home communication technologies on the other, SWoT framework supports exclusively web based technologies and in this aspect is more future oriented. While smart home gateway is a solution deployed in the home, SWoT framework is location independent and may be deployed in the home or cloud. Smart home gateway proxies all communication, while SWoT framework enables direct interaction between cloud services and devices. In development of access control mechanism for smart home gateway Kim et al. consider use case in which users grant access permissions to other users. In our work we consider more demanding use case scenario in which access permissions are delegated to software agents acting on behalf of particular user.

These fundamental differences reflect in corresponding solutions. For example, in this thesis there is a strong emphasis on realizing basic security infrastructure and determining the implications of this security infrastructure on cloud and device services implementation. The higher level concepts which build upon on this infrastructural solution, such as development of application specific policy model, or use of ontologies is left for future work. Compared to work of Kim et al. our solution is incomplete. However, we explore more thoroughly particular problem aspect - security architecture.

Service centric platforms, such as Xively [35], Paraimpu [36][37], ThingSpeak [38], and others share problem domain with SWoT framework, but adopt service centric design approach (single platform instance). They offer various functionalities, and capabilities, such as creating ad hoc mashups to direct device output to social networks, or to invoke action on the particular device based on the input from user defined source, such as Twitter feed. However, all the configuration is performed manually by users. In addition, users define semantics of APIs exposed on the devices, as well as of their device resource representations exposed on the platform. As resources and their semantics vary between different users development of standardized applications is difficult. Their current operation more closely resembles the function of data broker than the platform which could abstract device APIs and enable creation of standardized mashup applications.

Since these platforms realize less complex use case scenario than one envisaged for SWoT framework, security mechanisms which these platforms adopt are not capable to satisfy all security requirements present in SWoT

framework. In addition, the inner workings of these platforms' access control systems have not been publicly documented as they are proprietary solutions. While the most of the security features that these platforms offer is not directly applicable in SWoT, they way how they handle device provisioning and initial association of the devices with the platform (also known as bootstrap process) is relevant for SWoT [39]. Even though this particular problem aspect has been out of scope for this thesis project, we have considered it partially in order to better understand requirements for dynamic creation of authorization context.

Table 2.1: High level SWoT Threat Model.

ID	Threat	Attacker's benefit	Impact	Likelihood	Mitigation approach	Mitigation
1(1)	Someone (RU, RP, third party) impersonates RU	Access to sensitive data or actuation capabilities	High	High	Te	Authentication
2(13)	Someone (RU, RP, third party) impersonates RP	Collateral benefit from misleading FP and RUs	Med	Med	Te	Authentication
3(5)	Someone impersonates Framework Provider	Access to sensitive data or actuation capabilities	High	High	Te	Authentication
4(7)	FP is accessed by unauthorized entities	Access to sensitive data	High	Med	Te, Le, Tr	Authorization, Legal measures to prevent circumvention, of authorization. FP part of TCB
5(15, 19,24)	RP is accessed by unauthorized entities	Access to sensitive data or actuation capabilities	High	Med	Te, Le, Tr	Authorization, Legal measures, QoS, Trust Management. RPs part of TCB
6(4)	Someone floods the FP with requests	Collateral benefit from denial of service	High	High	Te, Le	Accounting, Legal measures, DoS-aware design
7(4)	Someone floods the RP with requests	Collateral benefit from denial of service	Med	High	Te, Le	Accounting, Legal measures, DoS-aware design
8(12)	A RP mis-advertises its service	Collateral benefit from misleading FP and RUs	Med	Med	Le	Legal measures, QoS, Trust Management
9(20)	A RP produces erroneous data or by error executes/ not executes particular actuation action	Collateral benefit from misleading service	Med	Med	Le	Legal measures, QoS, Trust Management
10(27)	A RP falsifies accounting	Preventing detection of unauthorized access	High	Med	Le	Legal measures, QoS, Trust Management

Chapter 3

Study of security mechanisms

In this chapter we provide high level overview of different types of distributed systems and architectural approaches for handling security in these systems. Moreover, we describe basic security components, such as identity and access management, and security procedures, such as authentication, authorization, and delegation. In the rest of the chapter we analyse multiple security protocols and select those most suitable for realization of SWoT's authorization architecture.

3.1 Distributed systems and security architecture

Continuous trend in development of computing systems over recent decades has been the shift from the local, isolated, centralized systems towards network connected, collaborative, and distributed systems. Increasing systems interconnectedness enables realization of novel applications and businesses opportunities. However, this same trend burdens realization of security systems as it becomes more difficult to achieve desired security requirements. The primary reason is that in these systems functional and security (non-functional) requirements are conflicting. Functional requirements drive the need for seamless interaction, resource sharing, and more openness. On the other hand, security demands isolation and reducing exposure in order to preclude resource access by unauthorized entities. Therefore, design of distributed systems (such as SWoT) requires careful balancing of functional and non functional requirements. Otherwise, resulting system may be insecure - if functional requirements are prioritized or inflexible - if applied security mechanisms are too restrictive.

The most simple distributed system in which one party interacts over

the network with the single service providing resources is illustrated in Figure 3.1a. The distinctive characteristic of this type of distributed system is that the service handles all security requirements individually without any dependence on third party services. The inner security architecture for this type of services is depicted in Figure 3.2 while the description of depicted security components is provided in section 3.2. This security architecture is relatively simple. Characteristic of this architecture is that all system components are tightly integrated. Moreover, components are managed by the system authority in isolation from other systems. The security system interacts with the outside external parties only for the purpose of authentication. All other procedures which are responsibilities of authorization and accounting security components are performed within system's boundaries.

Previous security architecture is suitable for simple services provided by single provider. However, it is unsuitable in cases when the system in question is more complex, e.g. single service provider offers multiple services (such as the case illustrated in Figure 3.1b). Previous approach for handling security in this more complex scenario is inadequate as would lead to many inefficiencies and it would be necessary that service provider manages each service completely separately. The common approach for addressing security requirements in these more complex scenario is to extract and assign responsibilities common for multiple services to a single internal service within the system. For example, instead of each service, which addresses particular functional needs, containing full authentication system, they implement only small part of authentication system requirements while for the rest of the authentication system functionality they rely and delegate to the specifically designated authentication service. This architectural approach (illustrated in Figure 3.1b) may also be used for addressing other security aspects such as, authorization, and accounting. However, as we will see in study of security mechanisms, realizing authorization as a service is difficult as it is more tightly integrated system operation compared to authentication.

Further proliferation of services and service providers introduces the need for increased integration of services across different administrative domains (scenario illustrated in Figure 3.1c). Primary drivers behind this additional level of integration is an opportunity to realize novel applications and improve system usability for end users. External system integration introduces dependencies, but it also enables services to delegate non core responsibilities to other services. This delegation is desirable in security domain as it enables centralized system management and monitoring. In systems which are deeply integrated, boundaries between administrative domains become less visible.

An example of this type of distributed system architecture are ecosystems of services which rely on large identity providers (such as, Google, Facebook, or Twitter). Integration in these systems is primarily focused on addressing identity management and authentication requirements. An another example is SWoT system architecture in which each smart thing is consider a service in its own administrative domain. In this architecture services to be integrated rely on services provided by the SWoT framework instance to handle their authorization and accounting requirements in addition to authentication requirements as in the previous example.

Approaches for handling authentication and authorization in distributed systems have evolved over time. Two major alternatives for orchestrating security protocols are *explicit* and *implicit* model [40]. In literature these two models are also recognized as *pull* and *push* model respectively [41]. Explicit model is simpler while implicit model is more flexible and suitable for application in complex systems. In explicit model client is not aware of security service presence as the communication between service provider and security service is direct and hidden from the client. On the other hand, in implicit model client is aware of security service presence as the communication between service provider and security service involves indirect exchange of message via the client. All the security protocols which we survey in section 3.3 are realizations of the implicit model.

3.2 Security components and procedures

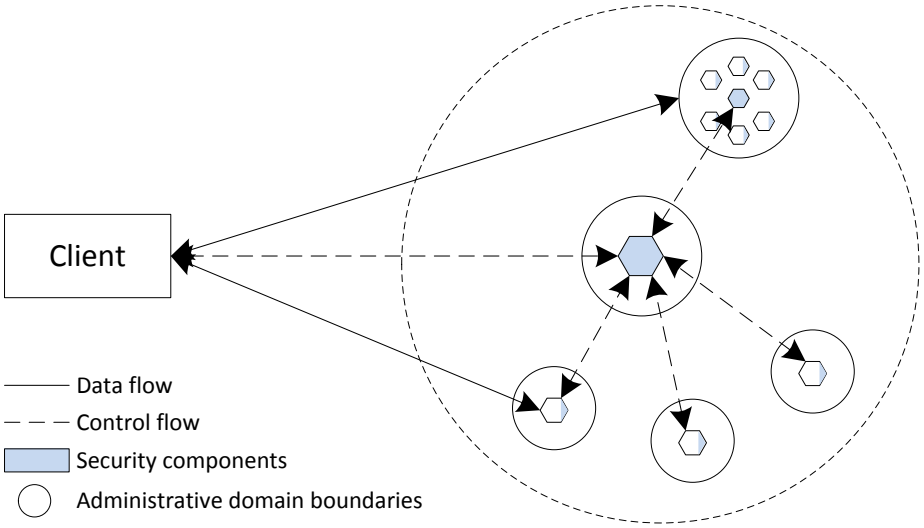
In security architectures two major components with distinct responsibilities can be recognized: identity management and access management component. These two components satisfy most of the system's security requirements. At the same time they are essential in orchestrating and handling security procedures, such as, registration, authentication, authorization, or delegation. We describe each component and relevant procedures in following section. Other security components exist (such as, accounting) and security procedures (such as, logging), but we do not discuss them as they are out of scope of this thesis project.

3.2.1 Identity management system

Secure identification of system entities is considered foundational in realization of security systems. This particular security function is the responsibility of identity management system. Identity management system is involved in registration and authentication of system entities. The notion

(a) Simple distributed system.

(b) Centralized distributed system.



(c) Federated distributed system.

Figure 3.1: Evolution of distributed systems.

of identity varies between different systems as it depends on particular system requirements and in this aspect it is similar to other security concepts that we describe here. For example, identity may be coarse grained and represent set of entities (such as organization, group of devices, etc.) or fine grained and indicate specific entity (such as particular user, software agent, or device). The design decisions adopted in realization of identity management system affect user's privacy. For example, system which requires users to use real names is often viewed as privacy invasive while system which allows use of pseudonyms is viewed as more privacy aware.

3.2.1.1 Registration

Registration represents the procedure for establishing trust between a client and a service provider. Moreover, it is a way to assign and associate system internal records with the specific third party entity (such as, human user, software agent, or physical device). Typically registration includes generation of new credentials which are later used to assert client's identity in authentication procedure.

One of the distinctive characteristics of distributed systems (such as those in Figure 3.1) is that the procedures and mechanisms for establishing identity are more complex compared to non-distributed systems. In non-distributed computing systems, such as desktop or mobile operating systems, managing and asserting identity is system provided feature and it is relatively simple to achieve. On the other hand, in distributed systems it is necessary to rely on relatively complex protocols and each service provider needs to provide necessary functionality in order to manage user's registration and later authentication.

Performing registration for each client service provider pair is inconvenient and not scalable especially in systems with large number of clients and services (such as, SWoT framework). These limitations are often avoided by adoption of mechanisms for brokering trust. One such widely adopted mechanism is a public key infrastructure through which a small set of trusted certificate authorities enables verification of user's identity without previous registration. An alternative approach, often adopted in consumer web services is to use services provided by large identity providers (such as Google, Facebook, or Github) for retrieving already established and sometimes even verified client's identity.

3.2.1.2 Authentication

Authentication represents the procedure for secure identification of system entities. This procedure relies on proofs of identity possession - credentials (such as password, secret key, certificate) which are established and issued in direct or indirect interaction of system entities with the identity management system during registration. In distributed systems (such as SWoT) authentication protocols specify how credentials are exchanged between the system performing authentication and the entity being authenticated. In addition, authentication protocols also define how exchanged credentials are verified. There are many different authentication protocols with characteristics that vary in many aspects. For example, different authentication protocols are suitable for different operating conditions, types, and characteristics of system entities, authentication system architectures, or desired levels of security.

In design of authentication part of security system the emphasis is often on components handling interaction with the external entities based on a particular protocol. However, effects of authentication have important impact on system's internal operation. Successful authentication typically results in instantiation of internal record which is entity's representation within the system. This record may be simple and contain single identifier or complex and contain other entity related information in addition to entity's internal identifier. This record is central artifact in enabling, tracing, and correlating entity's actions within the system. For example, in access control system the information from this record is used to locate and retrieve entity's rights while in accounting system it is used to log and subsequently monitor actions of particular entity. As potential failures in operation of authentication system could result in violation of system's security requirements, it is essential to guarantee its proper operation.

3.2.2 Access management system

Function of access management system is to enable authorized entities to access system resources and services in a way and an extent predefined by a system authority while at the same time preventing unauthorized entities to achieve the same. Access management system is central component of every system security architecture. Access management system cooperates with other security components (such as, identity management system) and distributed services to form fully operational security system. For example, access management system relies on inputs from the identity management system (such as, system's internal identifier for the entity requesting access).

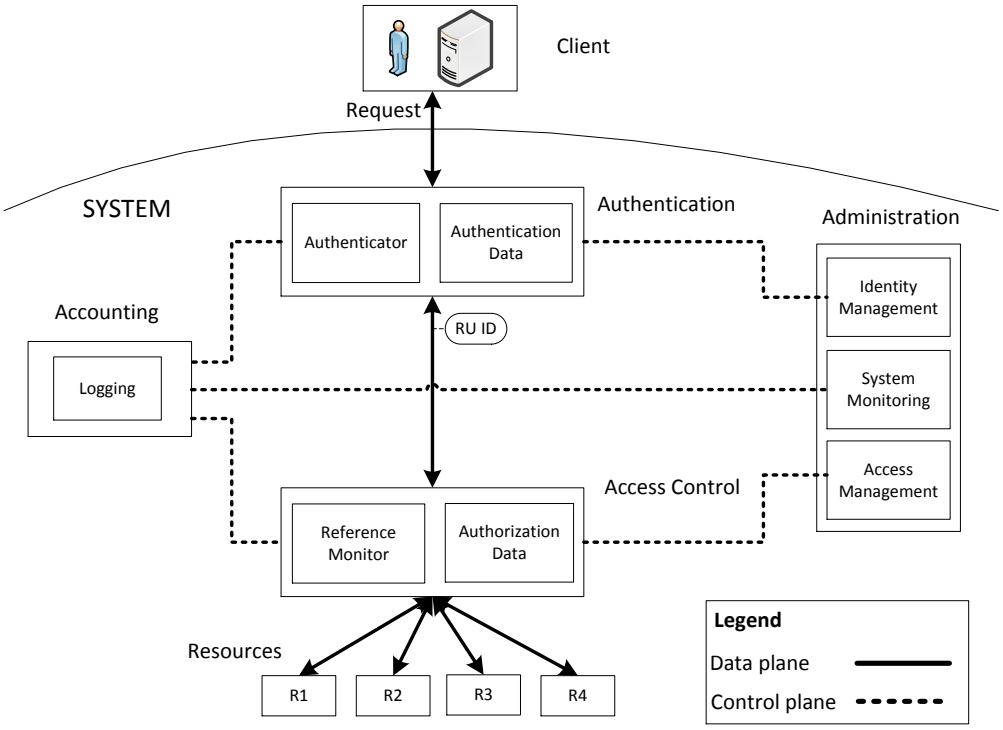


Figure 3.2: Security architecture.

Moreover, access management system interacts with accounting system and provides information about security events of interest (such as, successful or unsuccessful authorization). In addition, access management system provides functionality for administrators to manage overall system. For example, administrators are able to define access control attributes, policies, and assign them to chosen system entities.

3.2.2.1 Authorization

Authorization represents the procedure for determining if the already authenticated user is permitted to access a resource. It involves evaluation of properties associated with the user's account or some other artefact present in the system (such as, resource accessed, action performed, or environment conditions). For example, particular properties may indicate that the user is in specific role (such as, administrator, regular user, or guest), resource state (such as, confidential, secret, top secret), or environment conditions (such as, access requested from local or remote network). All these different properties

are referenced in authorization rules which are evaluated by authorization mechanisms.

When considering authorization it is necessary to analyse three important and closely related aspects. First, already discussed, is the access management aspect which is responsible for providing mechanisms for defining authorization rules, authorization properties, and for assigning them to designated system artefacts. The second aspect is responsible for authorization evaluation and it is typically addressed by adoption of particular access control model implementation. The third aspect is responsible for queering and distributing evaluation decisions. The importance of this last component is especially emphasized in distributed systems where procedures governing interactions between system entities may be complex. In these systems authorization protocols represent a class of solutions which address this particular aspect.

Access control models represent a class of solutions which provide theoretical and practical foundations for applying and implementing various authorization approaches. Different access control models have different properties and vary in complexity. For example, early access control models such as, access control matrix [42], access control list [43], and capability list are simple to implement but unsuitable for more complex systems. Role based access control model [44] is standardized [45], widely adopted, and applied in various systems. The attribute access control model through implementations such as, XACML [46], offers highest level of flexibility, but it is significantly more complex to use and manage.

Authorization protocols are a class of solutions for standardizing and handling interactions between system entities in a distributed systems which are performed for the purpose of querying and distributing authorizations. Authorization protocols exhibit different characteristics since requirements, assumptions, operating conditions, and goals for which they have been designed vary. Moreover, since non-functional interactions in distributed systems result in performance overhead it is common that all security interactions are addressed with a single solution. In addition, it is common that data exchanged in these interactions provides necessary input for both authentication and authorization systems.

All these three aspects of authorization are essential to realize fully functional authorization solution. However, in thesis project we focus on authorization protocols (their analysis and application in the SWoT platform) while we will consider the other two aspects in future work. This decision was motivated by the realization that authorization protocols and the system architecture to which they are applied represent the most challenging and least addressed aspect in SWoT project (in its current state).

3.2.2.2 Access control

Access control is a procedure for enforcing required security measures on a resource. Access controls allow or deny resource access based on inputs such as, information about the user requesting access and authorizations granted to the user, received from the authentication and authorization system respectively. In distributed systems, access controls implementation is tightly integrated with the functionality implementing security protocols. This is mainly due to reliance of access controls on data inputs from authentication and authorization services. Moreover, access controls are responsible for intercepting initial resource request and initiating authentication and authorization procedures.

Inadequate functioning and application of access controls may result in system security compromise. In this aspect they are similar to other security components. Access controls in web services are often implemented as integral part of various security frameworks. Implementation of access controls in security frameworks is often performed by developers with security expertise. Due to this errors in access control implementation are less common source of security vulnerabilities. However, access control placement and configuration are often performed by application developers with limited expertise in security. As a result, most security vulnerabilities are caused by access control misconfiguration or their inadequate application.

Access control term is often erroneously used interchangeably with the term authorization. However, access control and authorization are different concepts with different responsibilities. For example, while access control encompasses various technical measures and focuses on enforcing access decisions, authorization encompasses access control models, security protocols with authorization related interactions, and focusses on evaluation and distribution of access decisions.

3.2.2.3 Delegation

SWoT system can be described as a dynamic (entities become part of the system at runtime) and distributed (as entities are not physically collocated) system. In systems, such as SWoT, emphasis is on enabling collaboration and flexible service composition. Dynamic and flexible interactions require equally dynamic and flexible access control systems. One potential solution for realization of such systems is delegation. The concept of delegation in information technology domain has not yet been clearly defined and it is often used in various contexts and scopes. However, Pham et al. [47] define delegation as a “proxy process in which one entity makes available the

necessary rights to another entity to enable the receiver to perform certain duties on their behalf while addressing specified obligations and meeting particular constraints".

The application of delegation concept within particular system requires thorough understanding of system's characteristics. For example, for applying delegation in particular system it is necessary first to specify system boundary, the entities involved in delegation, and their relation. Moreover, it is necessary to define which rights should be delegated, how much it should be delegated, and for how long these rights should be delegated. To realize delegation on a technical level, it is necessary to choose particular implementation approach and appropriate mechanisms which are suitable for that approach. For example, delegation can be performed through exchange of tokens or by issuing policies [47]. Both approaches define procedures for issuing and honouring delegated rights as well as artefacts which are used to distribute those rights.

In systems in which delegation is applied most of the security system components have to be adapted to be able to support delegation and enable proper system operation. For example, access controls need to be adapted to properly handle delegation credentials. Logging mechanisms need to support checking who performed resource access and on behalf of whom. As a result, these security components are tightly coupled with delegation mechanisms and jointly share responsibilities. Due to interdependent nature of security components with delegation mechanisms, any improper handling of delegation within the system has a direct impact on overall security system operation. For more thorough discussion of delegation and its related aspects we refer interested readers to seminal work of Pham et al.[47].

3.3 Security mechanisms

While conceptually service integration from different systems is similar to service integration from single systems, requirements and challenges are distinctively different and require unique solutions. For example, security protocols' properties, such as standardization, flexibility, and security of service to service interactions are all more emphasized in case of service integration from multiple administrative domains.

In distributed systems the specialization of services for specific functions and operations increases importance of service interactions. In addition, this process also emphasises difficulties present in integrating distributed services, such as defining and standardizing interactions, message formats, and message content. Addressing these issues with non standardized solutions

or on an individual system basis is cost ineffective and it does not provide adequate results. For example, non standardized solutions do not undergo close scrutiny of field experts. Moreover, they require additional resources as common solutions do not exist and cannot be reused. Therefore, it is more common to rely on standardized security protocols such as those discussed in this section.

Security protocols which enable integration of services across administrative domains have to be flexible. This means that they need to support dynamic, runtime integration, and require minimal intervention from the system administrator. Security solutions which enable only design time integration and with large involvement of system administrators may be suitable for custom systems with small number of services. However, for systems such as SWoT, which is envisaged to enable integration of significant number of services, these more rigid security solutions are inadequate. Moreover, in case of SWoT there is an additional emphasis on security solution to be user friendly as administrators (end users) posses minimal knowledge from security domain.

Service to service integration across administration domains requires adequate mechanisms to be in place to protect communication confidentiality and integrity. Absence or improper configuration of these security mechanisms may invalidate security requirements and result in partial or full system compromise. Therefore, security protocols must feature built-in mechanisms for this purpose or rely on other widely adopted and well tested mechanisms specifically designed to provide communication confidentiality and integrity.

Security solutions applicable in service to service integration scenarios are often designed for particular set of technologies. For example, some solutions may be based on XML and Web services while other may adopt more modern technologies, such as JSON for its message encoding format and REST architectural style for designing and implementing services. It is desirable that security solutions are based on same set of technologies used for implementation of functional services. Otherwise, integration of these solutions can become cumbersome due to various interoperability issues. Moreover, requirement that application developers know two different sets of technologies may affect system adoption in developer community.

We have surveyed several prominent security solutions which are relevant for SWoT use case scenario. We present them and their characteristics in following material.

3.3.1 Kerberos

Kerberos is an authentication protocol initially developed at the Massachusetts Institute of Technology (MIT) in the 1980s. Through years it has gone through several revisions and its latest standardized version is Kerberos v5 from 2005 [48]. Kerberos is widely used solution in enterprise settings with many actors. In these systems Kerberos satisfies primarily authentication requirements and features limited support for authorization functions, such as support for communicating authorization privileges in standardized format.

Kerberos is a trusted third party authentication system which enables runtime introduction of Kerberos **clients** and **services** via Kerberos central entity, known as **key distribution center** (KDC) (detail description of the protocol is available in [48][40]). The Kerberos standard does not cover the process of establishing trust relation with the KDC as this particular system aspect is handled by system administrators during system design time.

The main underlying assumption in Kerberos design is that the attacker may read, modify and insert network packets at will [49]. Therefore, Kerberos protocol (depicted in Figure 3.3) has been designed to incorporate cryptographic mechanisms for authentication over untrusted communication network. However, Kerberos protocol provides only protection for the messages exchanged during authentication procedure. The communication between client and service which follows authentication is not protected by the standard application of Kerberos protocol.

Kerberos, similarly to all other third party authentication systems that we cover here, is centralized and represents single point of failure. If KDC for some reason is unavailable overall system that includes clients and services stops operating. Since KDC is core component of the system and one which all other entities trust, its compromise may result in complete system compromise. Therefore, as KDC represents high value attack target it is essential to be properly protected.

Kerberos is a protocol that relies directly on TCP and UDP transport protocols. Its design had and still has influence on design of modern security mechanisms. However, Kerberos has been designed before modern transport layer security mechanisms (such as TLS) and Web technologies (such as HTTP, HTTPS) have been even introduced. Therefore, it misses their characteristics and features. Due to all this, Kerberos application in Web based solutions is cumbersome and often inadequate.

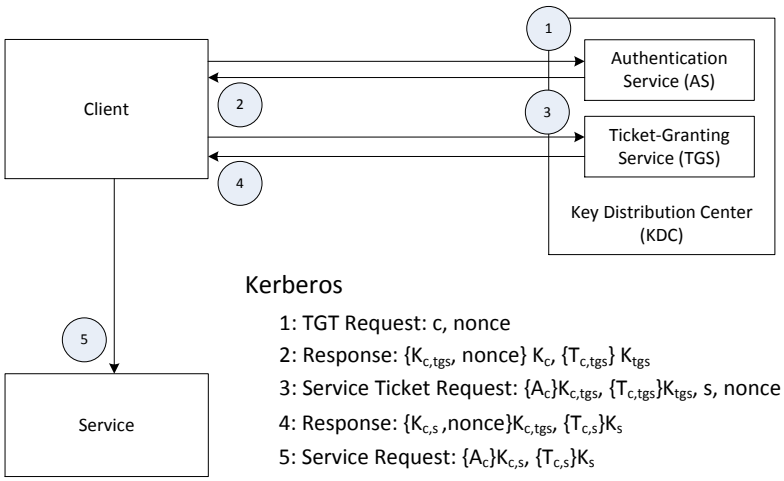


Figure 3.3: Kerberos (adapted from [40]).

3.3.2 Security Assertion Markup Language

Security Assertion Markup Language (SAML) is an OASIS standard and an XML based framework for exchanging authentication and authorization data. The latest version SAML 2.0 [50] has been introduced in 2005. SAML is a trusted third party authentication and authorization system. It is a popular solution for web based enterprise systems with complex authentication and authorization requirements. SAML is often integral part of security solutions in service oriented architecture system implementations.

SAML recognizes **subjects** (typically human user, but may also be a software agent), **services**, and central **SAML authority** (also known as identity, attribute, or authorization authority) as actors in SAML solution. SAML authority is the source of assertions which represent containers of authentication and authorization information (such as identity attributes, non-identity attributes, and authorization privileges). SAML assertions are exchanged within the system and consumed by services. SAML does not specify how subjects and services establish trust relation with the SAML authority. This particular issue is of out of scope of the standard as it is expected to be handled by system administrators during system design time.

Core part of the SAML standard is abstract generic solution which can be applied in different settings [50]. This part of the SAML standard defines *assertions* and *protocols*. For implementation of protocols using specific technologies (such as SOAP or HTTP) SAML defines different *bindings*. For

application in particular use case scenario SAML standard defines various *profiles*. Profiles facilitate SAML's feature selection and combination for SAML application in concrete use case scenario.

3.3.2.1 SAML analysis

SAML's main use cases which are of interest for this thesis project, are centralized authentication and centralized authorization [51]. Centralized authentication, also known as Single Sign On (SSO), enables the SAML authority to manage authentication of subjects on behalf of the services. SSO simplifies identity management at services side, improves user experience by obviating the need for subjects to authenticate at each service provider, and improves overall security as subject's credentials are shared only with the SAML authority. Centralized authorization enables the SAML authority to centrally manage authorization attributes (privileges, rights) of the subjects and to provide them to services on their request. SAML defines format and request/response protocols for communicating authorization attributes. However, SAML does not define how actual attribute values are specified or evaluated.

SAML by itself does not provide protection for communication between SAML actors. As a result, SAML standard recommends usage of transport layer security mechanisms [52] (such as TLS) to maintain confidentiality and integrity of communicated messages. SAML defines comprehensive set of features and mechanisms required for different use case scenarios and different technologies. This SAML's characteristic is its strength, as every feature required is often already available, and its limitation at the same time, as SAML solutions are complex to learn, configure, and manage.

3.3.3 XACML

eXtensible Access Control Markup Language (XACML) is an OASIS standard [46]. It is XML based language for specifying access control policies and encoding access decision requests and responses. In addition, XACML specification describes an abstract authorization architecture.

XACML architecture (illustrated in Figure 3.4) features several components with distinct functions. *Policy Enforcement Point (PEP)* is guardian of protected resources. It intercepts resource requests and sends corresponding access decision request to the PDP. Based on the content of access decision response PEP allows or denies access to the specific resource. *Policy Desion Point (PDP)* is a focal component in XACML architecture. PDP processes access decision requests coming from the PEP. It retrieves required

policies and attributes. PDP also evaluates policies and returns evaluation results to the PEP. *Policy Administration Point (PAP)* is a component responsible for storing access control policies and providing functionality for their retrieval. Mechanisms for managing policies vary between different PAP implementations. *Policy Information Point (PIP)* manages storage and retrieval of attributes associated with the subject, action, resource, and environment. Mechanisms for managing attributes also vary between different PIP implementations. *Context handler* abstracts implementation details of different components and enables loose coupling. For example, it facilitates decoupling of PEP - PDP and PDP - PIP components. In XACML implementations this component is a layer around PDP's core functionality (e.g. SunXACML [53]).

Normal operation of XACML based system features many interactions between different system components. PAP manages policies and makes them available to PDP before any access request is received (step 1 in Figure 3.4). Access requester sends resource request which PEP intercepts (step 2). PEP authenticates the requester and from the content of the resource request extracts required attributes (such as subject identifier, resource identifier, action identifier, etc). Then, PEP sends an access request with the attributes, encoded in a predetermined format, to the context handler (step 3). The context handler processes access request, builds XACML request context and sends it to the PDP (step 4). PDP requests from the context handler additional attributes that it requires for processing access request (step 5). The context handler in interaction with PIP retrieves required subject, resource, action, and environment attributes (steps 6 to 8) and makes them available to the PDP (step 9). The PDP evaluates policy based on received attributes and returns response context (containing access decision) to the context handler (step 10). The context handler processes request context and sends access decision response to the PEP (step 11). The PEP fulfils obligations present in access decision response (step 12) and depending on the result of access request evaluation provided in the access decision response permits or denies resource access (not shown).

3.3.3.1 XACML analysis

XACML has many **strengths**. XACML is *standardized* which enables development of interoperable implementations and tools. XACML has strong community with many business entities offering XACML based products. XACML is also a *generic* solution. It has been applied in systems belonging to different sectors, such as health care, education, financial, military, etc. XACML architecture is *distributed* and *loosely coupled*. As a result it

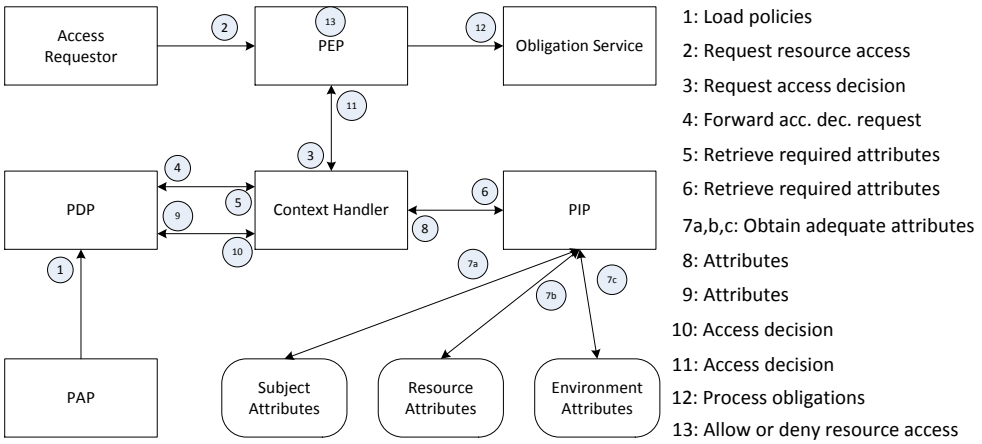


Figure 3.4: The XACML data flow, adapted from [46].

is extensible and very flexible. For example, policies or attributes can be retrieved in myriad ways. Moreover, XACML language is extremely *expressive*. In addition to rich set of elements defined in the language there are also extension mechanism for defining new elements. XACML has many *extensions* (such as SAML profile, RBAC profile, etc.) which enhance XACML with the missing functionality.

XACML has its **limitations** also. For example, XACML does not define transport protocol for PEP - PDP interaction. XACML does not specify how PEP requests should be authenticated by the PDP, nor how to handle case when there are multiple PEPs which are not equally trusted by the PDP. Note that these issues are present only when components of XACML architecture are distributed and not collocated as part of a single monolithic application. XACML language syntax is extremely complex which precludes direct modification of access policies. Publicly available XACML policy editors [54][55] do not improve user experience significantly.

3.3.4 OAuth

The OAuth is an authorization framework which enables third party applications to obtain restricted set of privileges to access resources hosted at particular service through a set of well defined interactions. OAuth 1.1 [56] and OAuth 2.0 [57] are two versions of OAuth protocol currently in wide use. While these two protocol versions are suited for same use case scenarios, their inner workings differ in large extent. In this thesis we consider exclusively

OAuth 2.0, to which we refer simply as OAuth.

Main actors in OAuth are: a **client** which is a generic third party application accessing resources provided by services; **authorization server (AS)** which is a central entity and a software agent responsible for managing different kinds of credentials, credential distribution to clients, as well as credential validation upon request from resource servers; **resource server (RS)** which is a software agent that in cooperation with AS guards access to resources hosted by particular service; **resource owner** a user which owns resources and is responsible for delegating its own resource access permissions to clients.

Main artefacts in OAuth are credentials of different types and scopes which represent OAuth specific form for communicating granted permissions. The OAuth features two types of credentials **authorization grant** and **access token**. Authorization grant is a temporary credential issued to the client by the AS upon user consent to grant particular permissions. The only purpose of authorization grant credential is to be exchanged for access token credential. Access token is time constrained credential, that may be implemented in a form of a handle or self contained assertion. It is used to indicate the context in which particular client interacts with service as well as the permissions granted to the client within that context's scope.

OAuth classifies clients as **confidential clients** which are capable of keeping credentials secure due to the way they are implemented (such as server based client with robust security mechanisms restricting access to credentials only to the client) and **public clients** which are incapable of maintaining confidentiality of their client credentials. Moreover, OAuth specification considers web application, browser based application and native application as different client profiles due to their inherently different characteristics and consequently security requirements. For example, web applications are considered to be confidential clients while browser based and native application are considered public clients as confidentiality of their credentials cannot be guaranteed.

OAuth defines multiple protocol flows (authorization grant types) which are accustomed to different client types (confidential or public), client profiles (web, browser, or native application), and use case scenarios. These are: authorization code, implicit, resource owner password credentials, and client credentials. Each protocol flow contains slightly different set of interactions resulting in four variations of OAuth protocol. However, one of interest to us in this thesis project is authorization code flow which is designed for web applications considered to be confidential clients. The authorization code protocol flow consists of three different interactions, depicted in Figure 3.5. First interaction represents request response exchange via user's browser in

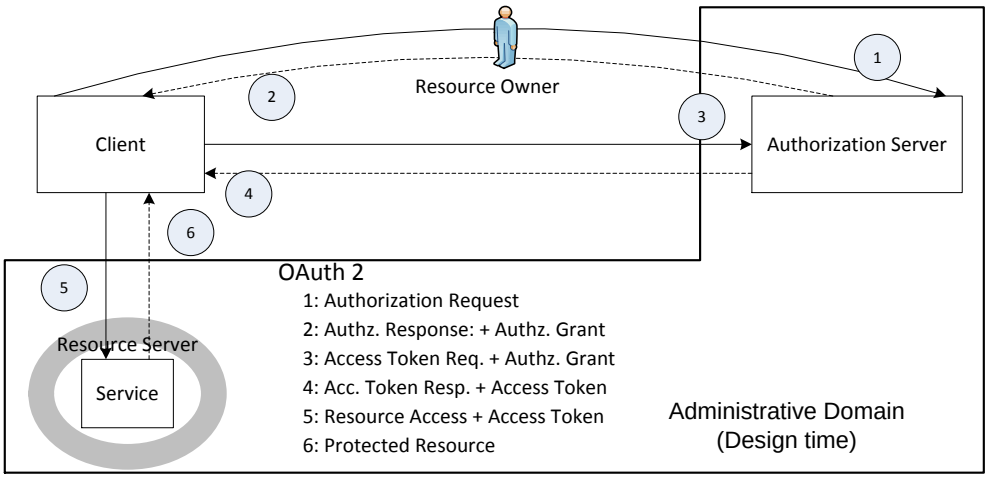


Figure 3.5: OAuth 2 authorization code protocol.

which client asks for particular permission and receives authorization code from the AS upon resource owner consent (steps 1 and 2). Second interaction is performed directly between the client and the authorization server. During this interaction client exchanges authorization code for adequate access token (steps 3 and 4). In third interaction client accesses service protected by resource server OAuth component (step 5). Client provides access token as a assertion of permissions that it possesses. If permissions associated with the token are sufficient to access resource, service responds to the client request with adequate response (step 6).

3.3.4.1 OAuth analysis

OAuth protocol addresses authentication and authorization system requirements as previously described security protocols. However, the distinctive characteristic of OAuth protocol is the introduction of human user, in role of a resource owner, in protocol operation. The OAuth through security concept known as delegation, in which resource owner plays significant role, addresses resource administration requirements. One part of this delegation process is runtime creation of user specific context. User context represents unification of distributed entities (client, service provider, and resource owner) into single, unique, logical domain. The other part of the delegation process relates to the credentials required for interaction with particular user context. These credentials are managed (created, updated, removed) by the service provider and used by clients so that interactions between entities and

consequently their effects may be associated and applied within particular user context.

OAuth protocol support for runtime creation of an authorization context is OAuth's distinctive feature. Other often considered positive characteristics of OAuth are its relative simplicity and design that relies on specific Web technologies which are currently considered best practices in design of web based solutions (such as RESTful design, JSON encoding format, adoption of TLS for transport layer security, etc.). Due to these characteristics OAuth is widely adopted. OAuth is also viewed as one of the main enablers of recent proliferation of Web APIs [58][59].

While OAuth is comprehensive security solution it does not address all security requirements in systems for which assumptions adopted in OAuth design do not apply. For example, while OAuth addresses the problem aspect of dynamically establishing trust between clients and central party, it does not cover the problem aspect of establishing trust between services and central party. This is due to the assumption that the services will be configured to trust central party at design time as they are implemented and managed by the same business entity. Moreover, the approach used in OAuth for addressing authorization requirements is based on the assumption that the nature and type of resources to be protected is known at the design time. This is reflected in implementation of OAuth artifacts known as scopes (privileges) which are determined at design time, and typically coarse grained. In addition, OAuth specification does not consider how in this distributed system central party gets updated with changes related to the resources (such as introduction of new resource on the service or its deletion).

3.3.5 User Managed Access Profile of OAuth 2.0

User-Managed Access (UMA) profile of OAuth 2.0 [60] is a novel solution for user centric access control. UMA has initially been proposed and developed by Machulak et al. [61][62] with the goal to design solution for addressing authorization requirements in systems with distributed services which have common trait that the user is the party responsible for administering resource access. Currently, the UMA work group [63], which is part of Kantara Iniative [64], oversees UMA development. UMA specification at the moment is in draft stage and represents work in progress.

UMA incorporates and augments OAuth with artefacts and interactions which enable realization of systems that feature dynamic (runtime) introduction of services in addition to dynamic (runtime) introduction of clients addressed by plain OAuth. UMA achieves this by considering services which provide resources as OAuth clients regarding their interaction with

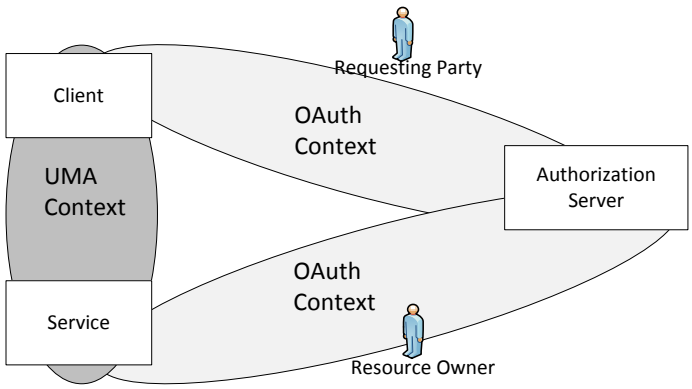


Figure 3.6: UMA and OAuth contexts.

the authorization server (see Figure 3.6). OAuth does not consider this particular interaction segment at all. Moreover, UMA overlays OAuth protocol with additional interactions between system actors which enable realization of authentication, authorization, and administration system requirements. There are many other differences which depict flexibility and consequently complexity of UMA when compared to OAuth. For example, in OAuth authorization server and services which provide resources are typically developed and managed by same business entity. In UMA the relation between services providing resources and authorization server is established by the resource owner at runtime. Therefore, they may or may not be developed by same business entity. While OAuth supports only protection of resources which are known at design time, UMA supports protection of resources which are known at design time or introduced at run time. All these fundamental differences lead to a radically different solution. Therefore, UMA designation as "profile of OAuth" may be slightly misleading.

UMA specification reuses terms defined in OAuth to refer to UMA actors and artefacts even though these do not match in many aspects. Due to this readers should be careful not to equate them. UMA **client** is an enhanced version of an OAuth client which accesses resources provided by services. UMA **authorization server** is an OAuth protected service with a specific function to manage and orchestrate UMA interactions. UMA **resource server** in cooperation with authorization server guards access to resources hosted and provided by the particular service. Even though its high level function is similar to OAuth resource server, UMA resource server realization

incorporates functionality for UMA specific interactions. UMA **resource owner** similarly to the OAuth is the owner of resources and the entity responsible for their administration. Resource owner may dynamically add new UMA compatible services to his/her domain. These services may or may not be developed by the same entity as the authorization server. In UMA person (natural or legal) may also act in **requesting party** role, in which it is responsible for brokering establishment of client - authorization server OAuth context. Simpler and more specific case is when person which acts as requesting party is the same person acting as resource owner. More generic and at the same more complex use case is when persons in these two roles differ.

The most important new artefact in UMA is the **requesting party token** credential. Requesting party token indicates the UMA context in which interactions between client and the service are performed (see Figure 3.6). It is also a reference handle to a record which contains permissions granted to the client acting in requesting party OAuth context for service acting in resource owner OAuth context. Moreover, UMA introduces **ticket** artefact which represents opaque handle to authorization server hosted record with permission requested but not yet claimed. UMA covers aspects of dynamic introduction of services into the system. For the realization of this particular system aspect UMA defines following artefacts: **resource set**, **scope**, and **permission**. Resource set represents service defined, smallest logical grouping of access-restricted resources. Moreover, resource set is an unit of management in UMA system. Scope is a service defined, logical operation applicable to particular resource set. Multiple different scopes may be defined for specific resource set (e.g. read, write, and manage scope). A permission is scope granted and associated with the client acting on behalf of specific requesting party.

Distinct procedures in UMA based system operation are: establishment of resource owner OAuth context, resource set registration and delegation of resource protection to the authorization server based on interactions in previously established resource owner context, establishment of requesting party OAuth context, and UMA core protocol with resource access and all other supporting interactions which are performed in resource owner and requesting party OAuth context. For establishment of resource owner and requesting party OAuth contexts UMA relies on plain OAuth protocol. For resource set registration UMA adopts OAuth resource set registration protocol [65].

UMA core protocol, depicted in Figure 3.7, includes following interactions which support resource access interaction: 1) issuing of requesting party token (steps 3 and 4 in Figure 3.7); 2) verification of requesting party token

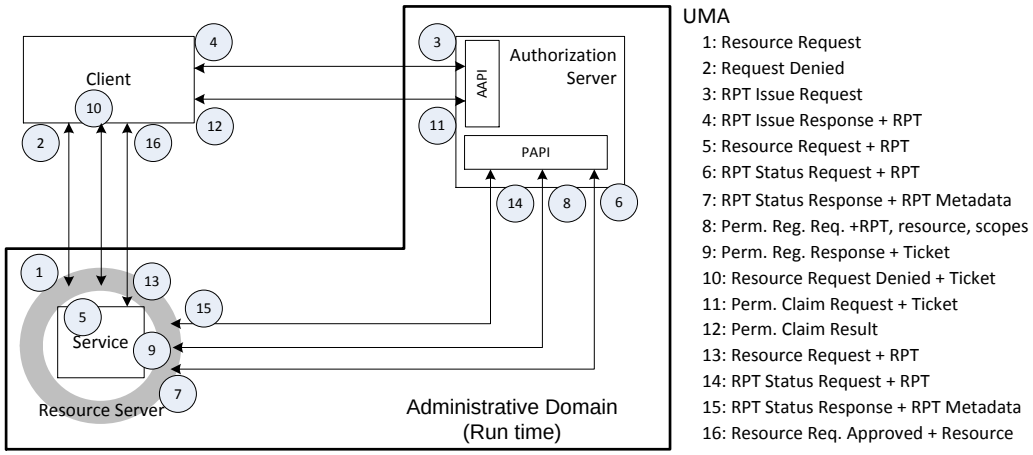


Figure 3.7: UMA protocol.

(steps 6 and 7); 3) permission request registration (steps 8 and 9); and 4) permission claim (steps 11 and 12). UMA core protocol without permission request registration and permission claim interactions enables protection of resources based on statically granted permissions and in this respect is similar to other previously described protocols. However, with permission request registration and permission claim interactions clients may obtain dynamically new permissions without user involvement which significantly increases system flexibility.

The core component of UMA is the authorization server which implements server side functionality for previously introduced interactions. Moreover, it provides protection API (PAPI) meant to be used by services and authorization API (AAPI) meant to be used by clients. PAPI contains OAuth protected permission registration endpoint and token verification endpoint. AAPI contains OAuth protected requesting party token issuance endpoint and permission claim endpoint. Tokens that grant access to PAPI and AAPI are OAuth access tokens named protection API Token (PAT) and authorization API Token (AAT) respectively.

We have provided here only high level overview of UMA core protocol and authorization server component. More detail description is available in Chapter 4 and UMA specification [60]. UMA incorporates other protocols and extensions (such as dynamic client registration protocol [66] and requesting party claim retrieval protocol [60]), but we do not discuss them as they are out of scope of this thesis project.

3.3.5.1 UMA analysis

UMA protocol is still in early stage of development. However, it features capabilities which make it suitable for adoption in systems such as, SWoT. For example, UMA protocol enables dynamic (runtime) integration of services. Moreover, administration efforts required to manage UMA based system are relatively small as features enabling dynamic integration are built into the protocol.

UMA is by design an authorization as a service solution. It provides comprehensive set of features for handling authentication and authorization requirements in a distributed system that spans across multiple administration domains. In addition, through set of related specifications other non-core features are also addressed. This wide coverage of issues by UMA related specifications facilitates implementation efforts and enables interoperability of UMA based solutions.

UMA adopts same Web technologies as OAuth (REST services, JSON encoding format, TLS for protecting confidentiality and integrity of communication, etc.). This facilitates integration with already developed OAuth solutions as well as their reuse for realization of comprehensive UMA frameworks. Moreover, it reduces effort in understanding and applying UMA as many web developers are familiar with these technologies.

3.4 Security mechanisms analysis

The Kerberos protocol is an inadequate mechanism for the SWoT authorization architecture for several reasons: it addresses only authentication requirements, it is a low level solution incompatible with web based technologies on which SWoT framework relies, does not support runtime registration of service providers, nor it is envisaged to be administered by non expert users.

SAML addresses both authentication and authorization requirements. However, it is a comprehensive solution for enterprise environments. Its complexity is a limitation in SWoT context, as it hinders development of third party applications. Moreover, SAML does not support runtime registration of third party service providers, nor it is a user friendly solution. In addition, SAML relies on Web services, SOAP, and XML, while SWoT framework adopts REST services, HTTP, and JSON as basic technologies. Due to these incompatibilities we do not consider SAML to be suitable solution for SWoT's authorization architecture.

XACML is a suitable solution for implementing authorization evalua-

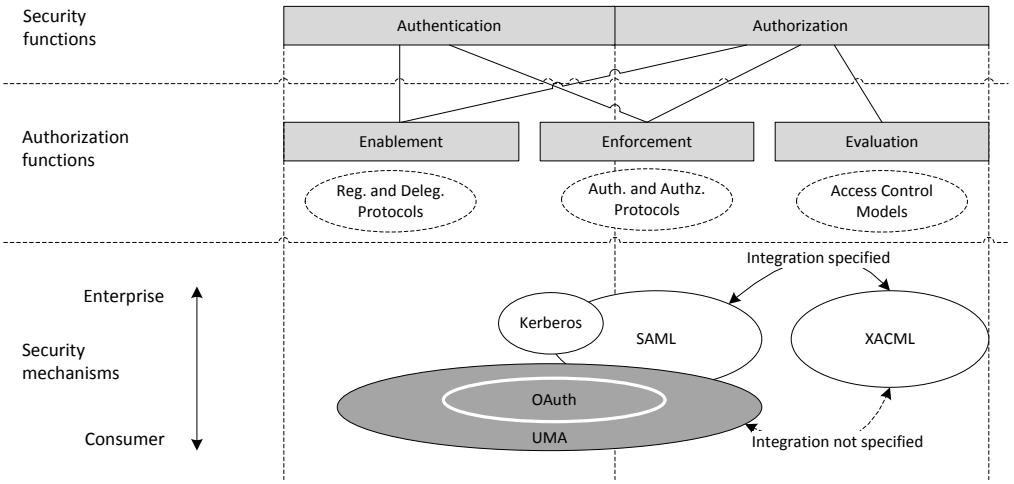


Figure 3.8: High level comparison of security mechanisms.

tion systems. However, XACML does not define transport protocol nor interactions between PEPs and PDP (service providers and authorization server respectively). This XACML limitation prevents its application in environments with third party service providers, such as SWoT. In enterprise systems, XACML is often coupled with the SAML as these two mechanisms address complementary aspects of the authorization architecture. However, even combined they do not satisfy SWoT requirements due to previously described SAML's limitations.

In our use case scenario for protecting resources provided by platform services we have chosen OAuth security mechanism. We know at design time which resources SWoT framework hosts and we can define adequate permissions (OAuth scopes). We expect that framework's services will be collocated. Due to this, lack of OAuth specification for an authorization server - services communication does not represent an issue. However, for protecting resources at third party resource providers OAuth is inadequate solution as we cannot define permissions for them at design time as we do not know their characteristics at design time. Moreover, lack of OAuth specification for authorization server - services communication in this case represents an issue as interaction between entities from different domains requires standardized solution.

In our use case scenario for protecting resources provided by third party resource providers we have chosen UMA security mechanism. UMA is designed to enable a centralized authorization service to protect resources hosted by third party resource providers. UMA with related specifications

defines runtime registration of resources and permissions applicable to these resources. This information is essential for enabling authorization service to offer to users resource access administration functionality. UMA also specifies the protocol for interaction between the authorization server and third party resource providers, as well as number of other extensions which are required for fulfilling this more complex use case scenario.

OAuth and UMA protocol are responsible for interactions in which resource servers query authorization server if particular client can access resource and receive response from the authorization server with the corresponding decision. However, OAuth and UMA are not responsible for evaluating access decision requests based on access control policies defined by the user. This is the responsibility of access control models (see Figure 3.8). Study, implementation, and integration of a suitable access control model with the UMA authorization protocol is left for future work.

Table 3.1: Characteristics of selected security mechanisms.

Characteristics/Mechanism	**Kerberos [48]**	**SAML [50]**	**XACML [46]**	**OAuth [57]**	**UMA [60]**
System authority	KDC	SAML Authority	PDP	Authz. Server	Authz. Server
Resource user	Client	Subject	Subject	Client	Client
Resource provider	Service Provider	Service Provider	PEP	Resource Server	Resource Server
Focus	Authentication	Authz. Enfor.	Authz. Eval.	Authz. Enfor.	Authz. Enfor.
Delegation	No	No	Yes	Yes	Yes
Runtime domain formation	–	–	–	Partial (RU)	Full (RU and RP)
Target domain	Enterprise	Enterprise	Enterprise	Enterp./Cons.	Enterp./Cons.
Confidentiality, Integrity	Built in	TLS	–	TLS	TLS
Transport Protocol	TCP or UDP	HTTP or SOAP	–	HTTP	HTTP
Data format	Binary (ASN1)	XML	XML	JSON	JSON

Chapter 4

System specification

In this chapter we provide a detailed description of all system interactions. We cover interactions defined by the UMA and OAuth protocol, as well as interactions protected by these security mechanisms. On a high level we divide all system interactions into three distinct phases of system's operation: *bootstrap phase*, *configuration phase*, and *operation phase*. Interactions from first two phases are not the primary focus of this thesis. However, they are crucial for understanding how our system (illustrated in Figure 5.1) operates. The third phase covers core system interactions.

4.1 Bootstrap phase

The bootstrap phase includes interactions for service (RLS and ALS) registration and initial authorization by which entities join SWoT domain in their role as RU or RP. We discuss service registration only partially, as it is out of scope of this thesis (see Chapter 3). However, we describe in detail service authorization by which they become part of SWoT domain. Bootstrap phase features exclusively interactions of services with the SWoT framework.

4.1.1 Service registration

Registration is a process during which SWoT for the first time establishes a relation with a particular service (ALS or RLS). In the process of service registration the service provides to the SWoT Framework its description (step 1 in Figure 4.2). This service description is an extensible set of various information items, such as service name in a user friendly form, short service description, icon, etc. This information is used in the SWoT Framework to

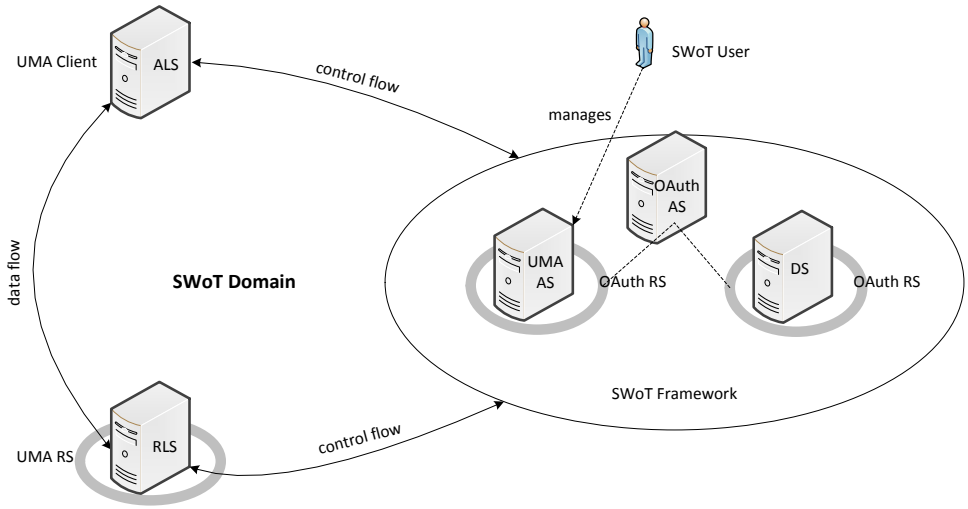


Figure 4.1: SWoT system.

enhance the user experience in various operations involving both ALSs and RLSs. In response to a registration request SWoT establishes an internal record and it generates an authentication credentials (e.g. client id and secret) that uniquely identify the service (step 2 in Figure 4.2). These credentials are then provided to the service (step 3 in Figure 4.2). Service is required to present authentication credentials in all future interactions with the SWoT Framework .

We recognize two alternative approaches for implementing this registration procedure in contemporary systems. In first approach, more common, this procedure is performed manually by a service developer. The developer enters the required information via a web form, based on the provided information the platform generates internal record and an authentication credential, developer retrieves the credential and embeds it into its service code. During this process the developer is also required to accept the platform's terms of use by which the developer is held accountable for service's operation in its interaction with the platform. This step is a mechanism for establishing trust between the platform and the service. Today popular Web platforms rely predominantly on this approach.

In second approach the process of registration consists of two separate sub-processes. In the first sub-process the service developer agrees to follow the platform's terms of use; the platform generates a bootstrap credential, which is required for second sub-process; and developer embeds

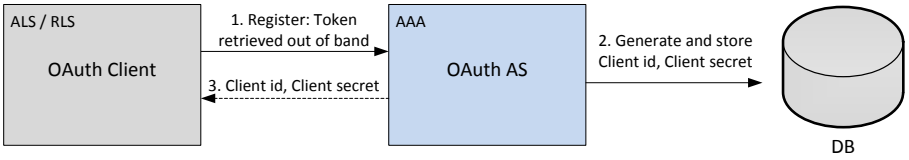


Figure 4.2: ALS and RLS Registration.

this credential into its service code. In second sub-process the service autonomously registers with the platform: the service requests registration and provides the bootstrap credential; the platform correlates this service with its developer using the bootstrap credential; if this correlation is successful then the registration request is satisfactory and the service receives in response a newly generated authentication credential. This approach enables realization of more advanced use cases. For example, multiple instances of the same service can register and configure themselves independently with unique authentication credentials. The first sub-process in this approach is platform specific and not well suited for standardization. However, second sub-process, named *dynamic client registration*, is focus of recent standardization efforts [67][66].

In the SWoT domain two types of services with different characteristics exist (see Chapter 2). For ALSs which are single instances and managed by the service developer both approaches are suitable but the first is used more often. For RLSs which are multi instance and managed by end users, the second registration approach is more adequate as each instance will obtain its own set of authentication credentials.

We have provided here only a general overview of the registration approaches, as concrete implementation mechanisms are out of the scope of this thesis. However, readers interested in concrete implementation mechanisms should consult the Generic Bootstrapping Architecture explained in Section 8.3.1 of SENSEI Security Architecture [12], Security Bootstrapping Solution for resource-constrained devices [68] and Xively provisioning API[39]. All of the previous mechanisms were directed towards scenarios focused on services with RLS characteristics, which are more challenging compared to those involving the ALS.

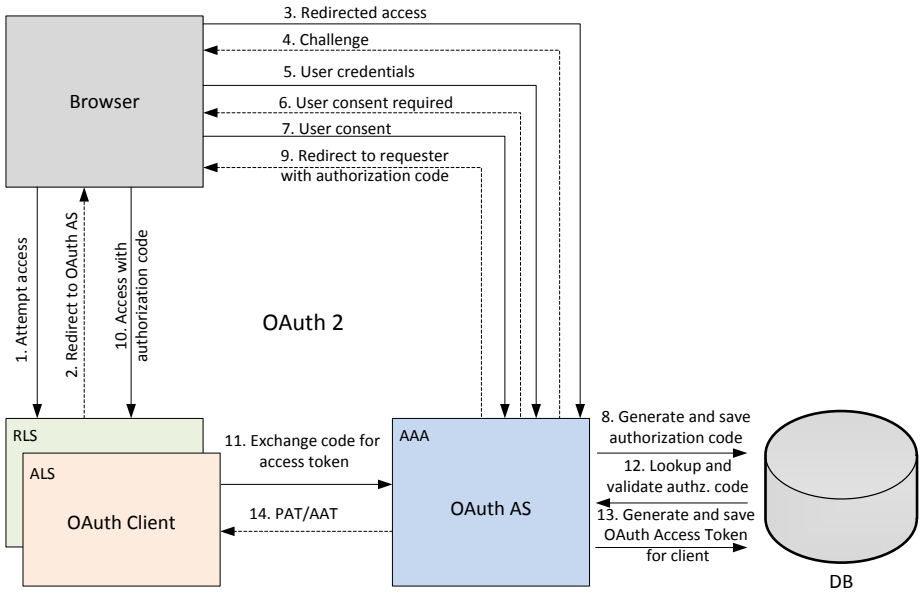


Figure 4.3: Association.

4.1.2 Service association

A prerequisite for a service (both ALS and RLS) being able to operate within a SWoT system is that they are registered. We have explained how this may be achieved in the previous subsection. However, possessing solely an identity is insufficient. The fact that service is entity who it claims to be, moreover one which SWoT system recognizes, does not automatically entitle it to operate within a SWoT system. What is necessary is that a SWoT user delegates the proper rights to it. Since there are two different roles (RU and RP) in the SWoT system there are also two rights that can be delegated. The first right entitles a service to act as RU, while the second entitles it to act as RP. We note readers that in this thesis we assume that an ALS will only act as an RU, while a RLS will only act as RP. We do not consider cases in which ALS acts as RP, RLS acts as RU, or an ALS or RLS act as both a RU and a RP. Study of these use case scenarios is left for future work.

In Chapter 3 we have identified that OAuth 2.0 is a suitable mechanism for delegating authorization rights. Moreover, that *authorization code* is a suitable authorization grant. Therefore we present here its application for our use case. Figure 4.3 depicts all interactions between the involved

entities, while bellow we describe each of them. Note that this is simplified explanation of OAuth 2.0 protocol (detailed explanation is available in [57]).

1. A SWoT and ALS user using a browser requests a particular resource from the ALS. To satisfy this request ALS needs to access user's resources at SWoT (e.g. those resources provided by the DS).
2. The ALS processes user's request and determines that it does not have the required credentials to interact with the SWoT's discovery service. To retrieve credentials it initiates the OAuth authorization code flow with a redirect response destined to the SWoT's OAuth AS. With this action the ALS requests the RU right.
3. The browser processes the response and follows the redirect to the SWoT's OAuth AS.
4. In order to delegate rights the user needs to be signed in at the SWoT. If this is the case, this and the following step are skipped. A common mechanism for user authentication in web applications is form based authentication. This process starts by the OAuth AS returning a response to the browser requesting the user's credentials.
5. The user enters the required information, e.g. user name and password, and submits the request. OAuth AS validates credentials and authenticates the user.
6. At this stage the user is authenticated and the OAuth AS starts processing the initial request originating from the ALS. OAuth AS returns a response to the user's browser requesting its consent to issue a RU right to the ALS.
7. The user submits its consent to the OAuth AS.
8. Now the OAuth AS generates an authorization code.
9. The OAuth AS redirects the user's browser back to the ALS. Redirect response contains an authorization code destined for the ALS.
10. The browser follows the redirect by issuing a resource request with all the necessary information received from the OAuth AS to the ALS.
11. The ALS processes the request and retrieves an authorization code from the request. Following the OAuth protocol the ALS sends this authorization code to the OAuth AS service.
12. The OAuth AS validates received authorization code.
13. If this validation is successful the OAuth AS generates an access token and internally associates with it a RU right. In UMA terminology

Authorization Access Token (AAT) is term used to refer to an access token with a RU right.

14. The OAuth AS in response to the ALS provides a AAT. The ALS saves the AAT and uses it in all subsequent interactions with the SWoT's services.

Through previously described set of interactions user delegates a right to the ALS to act as RU. The procedure in which SWoT user delegates right to the RLS to act as RP is almost identical to the previous one. The only difference is that a right which RLS requests and receives is a RP right. The OAuth AS in response to the RLS request issues the access token associated with the RP right, which in UMA terminology is known as a protection access token (PAT).

As an outcome of procedures in this phase we have a SWoT domain with registered an ALS and a RLS which are authorized to interact with the SWoT framework in RU and RP roles, respectively. In the following sections we describe operations that they perform in these roles.

4.2 Configuration phase

The configuration phase includes interactions between services, viewed at this stage as RPs and RUs, with the Discovery Service (DS) component of the SWoT framework. These interactions include *resource registration* and *resource discovery* in which a RP and a RU interact with a DS, respectively. We also describe SWoT's administration capabilities. Configuration phase, same as the bootstrap phase, does not include interactions between RUs and RPs.

We limit our discussion of a DS here to a high level overview. The intricacies and implementation details of a DS are out of the scope of this thesis. However, the protection of the DS from unauthorized access is within the scope of this thesis. Therefore, on example of DS we explain how our authorization solution protects SWoT's internal services.

4.2.1 Resource registration

The *resource registration* interaction is an essential interaction in the SWoT system. Through this interaction the SWoT framework becomes aware of the resources which third party resource providers host. In addition, the SWoT framework relies on information exchanged in resource registration interaction for performing resource management.

The RP interacts with DS to register resource descriptions (see Figure 4.4). This interaction consists of following steps:

1. SWoT user (RLS admin) using browser requests from RLS to register all its resources with SWoT. This step may be optional if the RLS performs this operation automatically, e.g. after joining the user's SWoT domain.
2. The RLS acting in the RP role submits resource descriptions to the DS.
3. The DS processes each request for resource registration and saves the resource descriptions.
4. The DS indicates the status of each registration request's processing in a response. In case this response indicates that the registration was successful, RLS starts delegating resource protection to SWoT(see Subsection 4.3).

The procedure for registering resource descriptions and the resource description format are defined by a UMA related specification [65]. However, in UMA context resource registration is not performed for the purpose of enabling later resource discovery, but for enabling a user to identify a particular resource and to specify access control policy for that same resource. As a result of this design characteristic resource description format is simple, but inadequate for applications with different requirements. For example, feature rich discovery service requires more expressive resource description format. Therefore, this problem aspect requires research into more appropriate resource description formats which would provide UMA required features and be able to support feature rich discovery service.

4.2.2 Resource discovery

The *resource discovery* interaction enables the RUs to discover at runtime the RPs and the resources which they provide. Through this interaction the SWoT framework facilitates interaction of RUs and RPs as well as realization of complex mash-ups.

The RU interacts with DS to obtain information about resources of interest (see Figure 4.4). This interaction consists of following steps:

6. SWoT user (ALS user) using browser requests from ALS to discover resources at SWoT relevant to its function. This step may be optional if the ALS performs this operation automatically, e.g. after joining the user's SWoT domain.

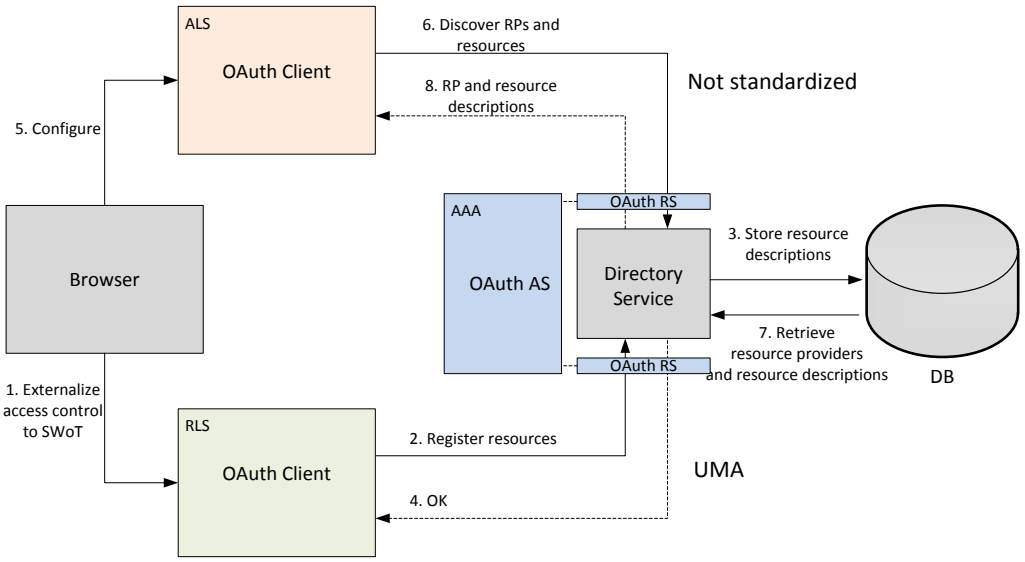


Figure 4.4: Resource registration and discovery.

7. The ALS acting in the RU role submits a resource discovery request to the SWoT's DS.
8. The DS processes requests and retrieves the matching resource descriptions as well as service descriptions of the RPs hosting them.
9. The DS returns to the ALS matching service/resource descriptions.

The DS provides information about resources and the RPs hosting them. This information may include the RP's endpoint, which a RU can use to directly interact with a RP. Moreover, DS may provide information such as the RP's name, icon, and description that could enable the RU to provide a better user experience. In contrast to the resource registration which is in process of standardization, resource discovery in the context of the SWoT project is currently not covered by any specification.

SWoT's services such as DS are protected by OAuth. This is depicted in Figure 4.4 where requests in steps 2 and 6 are intercepted by the OAuth RS and validated in cooperation with the OAuth AS. Depending on credentials provided and results of their validation OAuth RS allows or denies further processing of the particular request. For example, a RLS is required to provide valid PAT for its resource registration request to reach DS. Similarly, an ALS is required to provide valid AAT for its resource discovery request to be processed by DS.

4.2.3 Resource administration

In our solution we adopt XACML language for policy specification. XACML policies are expressions over four different classes of attributes including: subject, resource, action, and environment attributes. *Subjects* in our particular case are ALSs acting as RUs and relevant subject attributes express this. We envisage following attributes: service identifier, service developer, and service function for which system provides values, or attributes such as service trust level for which user defines values. *Resources* are what needs to be protected. There are many resource attributes as resources are heterogeneous and vary between different resource providers. In addition to those simple such as resource identifier we envisage following attributes: resource type (sensor, actuator etc.), resource provider identifier (fridge, oven, lamp 1, lamp 2), resource provider type (fridge, lamp, door lock), location of resource provider (home, kitchen, living room). *Actions* represent applicable operations for resources. In our system we use a single action attribute and that is an action identifier. Defining additional action attributes is possible but for our use case unnecessary. *Environment* attributes provide contextual information. Most prominent attributes of this class are time based attributes which are useful for defining access control policies with time based constraints. Their values are provided directly by the system when access control policies are evaluated.

We define a subject and a resource provider attribute values during the registration of the ALS and RLS at SWoT. For example, system assigns unique value to subject and resource provider identifier attributes. Values for other static attributes, such as a service type or service developer, can also be generated at the registration time. However, dynamic contextual attributes (such as, current location of resource provider) cannot be generated at the registration time and require additional communication between system entities. We define resource attributes and their corresponding values when they are registered at the DS. For example, system assigns a unique value to resource identifier attribute when that same resource is registered. Value for other resource attributes are extracted from the resource description. Resource registration also enables definition of action attributes. Actions and their identifiers are based on UMA scopes which are integral part of a resource description. Time attributes are not related to third party entities and therefore their values are managed internally. As they are dynamic their values are generated at runtime.

XACML based access control policies are extremely expressive. SWoT user can reference single or multiple attributes from different attribute classes when defining access control policies. By choosing particular attributes

and their values user can restrict and expand applicability of his policies. This means that user may have single policy which applies to all resources or multiple access policies which apply to different sets of resources. For example, SWoT user may define access policy which references particular resource identifier and therefore applies only to that resource. However, this approach is inefficient. Instead, user can define policy which applies to a set of resources by referencing an attribute which is shared between multiple resources. For example, if user wants to specify access control policy which would apply to all lamp devices, then it needs to reference a resource provider type attribute with the value lamp. User can also choose to leave out particular attributes and even whole attribute classes. We describe how system evaluates these policies in Section 4.3.3.

4.3 Operation phase

The operation phase includes *resource access* interaction between services (RUs and RPs), as well as all other interactions required to secure it. These interactions are *token retrieval*, *token verification*, *permission registration* and *permission retrieval* which are system external interactions and *permission evaluation* which is system internal interaction. SWoT framework, more precisely its UMA AS, has a pivotal role in all interactions during this phase.

4.3.1 Authentication

Process of authentication (illustrated in 4.5) consists of following steps:

1. The RU requests a resource from the RP. Since this is the initial interaction between the RU and the RP, the request does not contain the required credential (RPT).
2. At the RP a security component intercepts the request. It detects that request does not contain a RPT and returns a response indicating an error, with information in response that the RU can retrieve a valid RPT from the SWoT framework.
3. The RU detects via the error response that it needs an RPT for accessing the RP. To retrieve this RPT the RU sends a request to the SWoT's UMA AS indicating in the request for which RP a new RPT will be used. Since the RPT issuing service at UMA AS is OAuth protected the RU embeds in its request the required credentials: first a credential asserting the ALS's identity and the second credential (OAuth Access Token) asserting that it has the RU right.

4. The UMA AS processes this request for new RPT. It generates a internal record representing the relationship between a RU and a RP. RPT represents unique string token which indexes this internal record.
5. The UMA AS returns a response to the RU with the new RPT as part of the response.
6. The RU extracts the RPT and retries its access of the resource at the RP.
7. At the RP the security component detects that the request contains a RPT. However, it does not know if it is valid. To determine this RP requests the UMA AS to validate the RPT. Since the RPT validation service provided by the UMA AS is OAuth protected the RP embeds in its request the required credentials: first a credential asserting the RLS's identity and the second credential (OAuth Access Token) asserting that it has the RP right.
8. The UMA AS processes the request for RPT validation by performing two checks. First it checks that there is an internal record identified with this particular RPT and second that the RP specified in the record matches RP which requested validation. If both checks are successful, then the UMA AS retrieves a capability list (CL) associated with this RPT. Initially this CL is empty.
9. The UMA AS returns a response to RP indicating the result of the RPT validation. If the validation was successful the response also contains a CL.
10. At the RP the security component processes the response from the UMA AS. If validation is unsuccessful, then it responds as in step 2. If validation is successful, then it allows further processing of the RU request from step 6. At this stage the security component in the RP creates an internal record for the RU. The RPT identifies this record which contains a CL received from the UMA AS in the previous step. Depending on required authorization granularity at the RP we can have one of two different cases. In the first case when coarse grained authorization is implemented at the RP, then a valid RPT grants the RU access to all RP resources. This case is described here and depicted in Figure 4.5. In the second case the RU has to have a valid RPT and the CL associated with it needs to contain appropriate rights. This case requires additional interactions and we describe it in subsection 4.3.2.
11. As a result of successful authorization in the previous step the original request is processed.

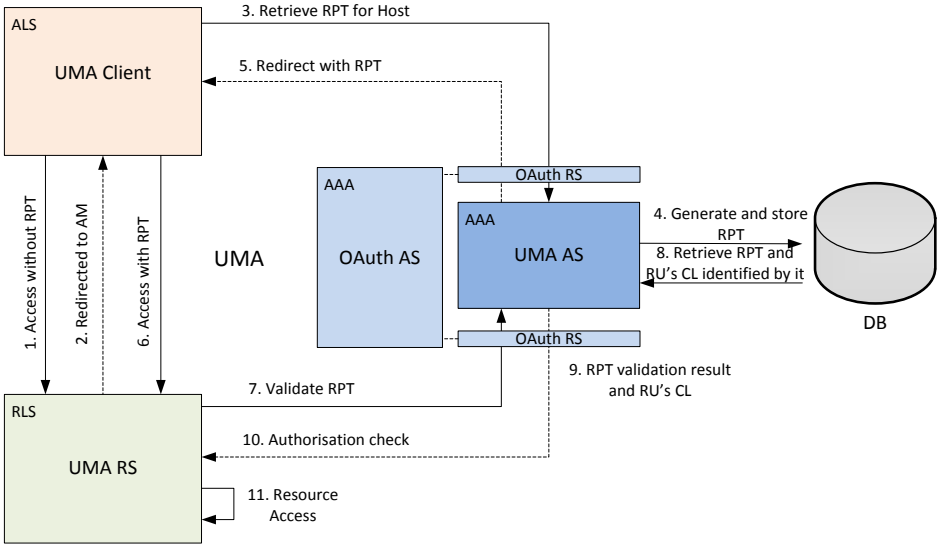


Figure 4.5: Authentication flow.

4.3.2 Authorization

When RU does not possess required permissions RP initiates permission obtaining flow which consists of interactions depicted in Figure 4.6. These interactions represent continuation of the process described in subsection 4.3.1 and include following steps:

10. The RP security component is preconfigured with information about rights which the RUs need to have to successfully access resources. Based on this information the RP performs an authorization check. It determines if a particular RU in the associated CL has the required rights. As we are continuing the process described in subsection 4.3.1 we know that the CL is initially empty. Therefore access is *not* granted.
11. The RP security component submits a request to the UMA AS in which it provides an RPT (which identifies the RU) and the list of rights that the RU is lacking for successful access to the resource.
12. The UMA AS performs the necessary checks and creates an internal record containing a RPT and a list of the rights received from the RP. It identifies this record using a unique string token, also known as **ticket** in UMA terminology.

13. The UMA AS returns a response to the RP indicating the result of request processing. If the processing was successful, then the response contains a ticket.
14. The RP extracts the ticket from the UMA AS's response and embeds it in response to the RU's request from step 6 indicating that resource access is denied.
15. The RU detects from the error response that it is missing some rights to access resource. It does not know which rights nor how many of them it needs. However, it knows that it can try to retrieve them by submitting a request to the UMA AS with a RPT and a ticket embedded in the request.
16. The UMA AS uses the ticket to retrieve the internal record with the missing rights and performs the necessary checks. The UMA AS uses retrieved information to populate the evaluation request.
17. The UMA AS relies on the implementation of the abstract request evaluator interface to evaluate if the RU can obtain the necessary rights. The UMA AS initiates evaluation process by sending a suitable evaluation request to the Request Evaluator component.
18. The Request Evaluator component evaluates the request and returns a response containing a decision.
19. If decision is to grant rights, then the UMA AS will add those rights to the RU's CL and store it. Otherwise, CL will not be modified and the RU will not receive any additional rights.
20. The UMA AS returns a response to the RU indicating the result of the request processing. If the result indicates failure, then the RU knows that its request for additional rights has been denied and that it cannot access the resource. However, if the result indicates success, then it knows that the missing rights have been granted to it and it continues the access procedure.
21. The RU retries accessing the resource at the RP. This request is identical to the request in step 6 in Figure 4.5.
22. Same as step 7 in Figure 4.5.
23. Similar to step 8 in Figure 4.5. The only difference is that CL is no longer empty, but contains some rights.
24. Same as step 9 in Figure 4.5.
25. The RP processes the response from the UMA AS. From the response which indicates that the RPT is valid the RP retrieves new CL. It

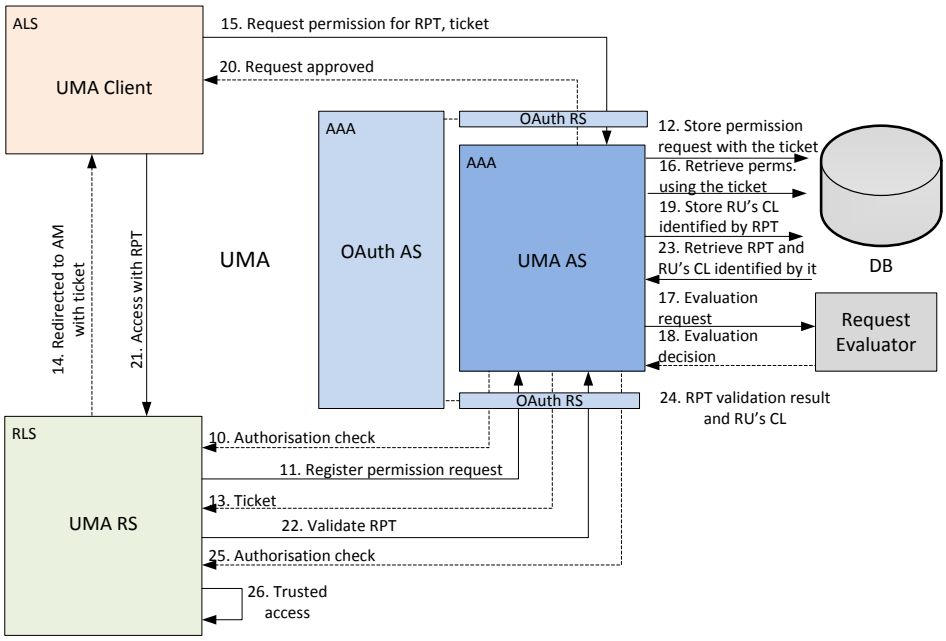


Figure 4.6: Authorization flow.

compares the rights required for accessing the resource with those present in the CL. If the CL contains all of the necessary rights, then the authorization succeeds, if this is not the case the authorization procedure restarts from step 10.

26. As a result of a successful authorization in the previous step the original request is processed.

4.3.3 Authorization evaluation

Authorization evaluation represents a set of internal system operations through which system issues new and invalidates old permissions granted to RUs (ALSs acting on behalf of specific user). Figure 4.7 depicts the internal interactions and those external which drive them. Two main artifacts are permission requested record and requesting party token record. First record contains permissions which are requested and are to be evaluated, while the second contains permissions that have been already granted. The permission requested record is generated when RLS registers permissions necessary to access particular resource on behalf of specific user (steps 11 and 12 in Figure 4.6). Permissions in the permission requested record are

evaluated on reception of the permission claim request from the ALS (step 15 in Figure 4.6). The internal system interaction flow (illustrated in Figure 4.7) features following system operations which combined perform permission request evaluation:

1. The UMA AS retrieves previously registered permission requested record from the database. From this record it retrieves necessary information, such as permissions requested, resource identifier, resource provider identifier, etc.

2. The UMA AS encodes this information into predetermined format and submits the access decision request to the concrete implementation of request evaluator.

3. The request evaluator processes request, retrieves necessary information from the request, and retrieves applicable access control policies from the database.

4. The request evaluator retrieves attributes referenced in applicable access control policies, performs access control policy evaluation, and derives access decision.

5. The request evaluator provides the access decision to the UMA AS in the access decision response.

6. The UMA AS parses the response and processes the access decision. If the access decision is to grant new permissions, then system associates new permissions with the corresponding requesting party token record. Otherwise, system does not modify requesting party token record.

When the system processes permission request it provides the result of the evaluation to the ALS. A success status in the response indicates that requested permissions have been granted (step 20 in Figure 4.6), while the response with a failure status indicates that permissions cannot be granted as resource's access control policy does not allow it. The new permissions granted to the RU, in the case of successful evaluation, are visible to the RLS on the next RPT validation request (step 22 in Figure 4.6).

Permissions granted to the RU are time limited, which forces RUs to go through permission evaluation process more frequently. This approach is realized with a parameter - permission expiration period, associated and stored with permissions in the requesting party token record. This particular parameter is provided in the CL sent to the RLS in response to RPT validation request. Within RLSs permission expiration period is

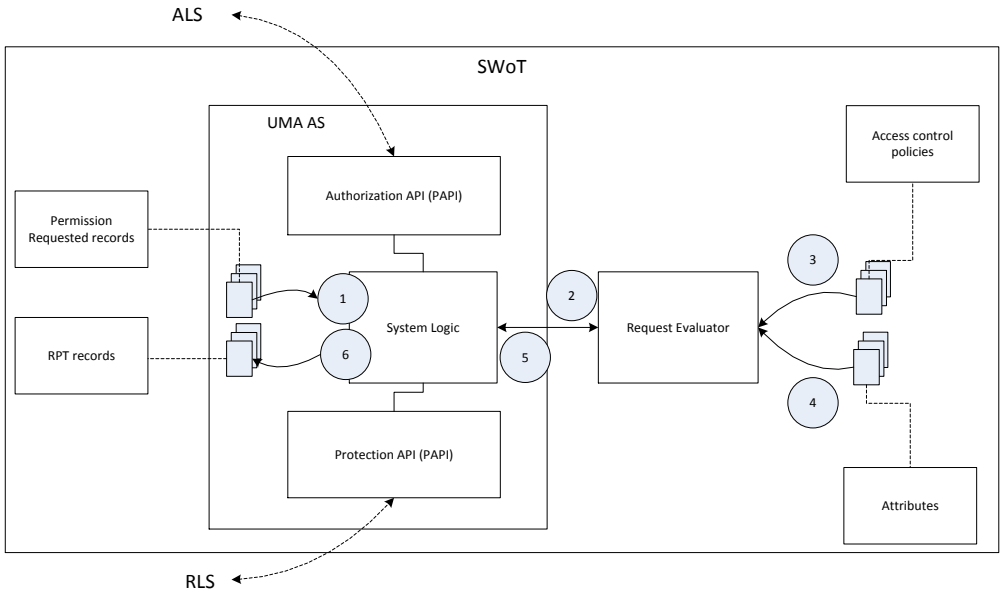


Figure 4.7: Permission request evaluation.

used as an input for caching mechanism. The permission expiration period assigned to granted permissions may affect overall system operation as with different permission expiration periods overall system behaves differently. For example, when permission expiration period is short system behaves as it is an implementation of dynamic access control. On the other hand, when permission expiration period is relatively long system behaves as a static access control system. Permission expiration period also affects system performance (for detailed discussion see Chapter 6).

Our overall system architecture resembles the classical XACML architecture described in Chapter 3. UMA RSs share characteristics of XACML PEP entities. UMA clients are similar to XACML access requesters. UMA authorization server provides functionality for orchestrating various authorization flows with which it enhances the request evaluator component which resembles XACML PDP.

Chapter 5

Implementation

UMA is novel protocol. As a result, very few implementations are currently available. UMA WG documentation [69] lists several implementation efforts out of which SMARTAM driven by UMA/j framework and Fraunhofer AISEC solutions stand out as most complete. SMARTAM project published prototype implementation of UMA client and resource server components [70]. However implementation of UMA authorization server, main component in UMA, has not been published. Fraunhofer UMA implementation has been open sourced in July 2012 and it is publicly available since then. At the time when we started our implementation effort these solutions were not available. Therefore we have implemented our own UMA framework to which we refer as Spring Security UMA (SSU) framework.

In rest of this section we explain how our SSU framework realizes UMA roles of authorization server, resource server, and client as well as where and why our implementation deviates from the draft UMA specification. Our implementation is based on Spring Security and Spring Security OAuth2 libraries. Therefore we first provide short introduction of these two solutions. We continue with discussion of SSU framework from high level architectural perspective. More detail discussion would require detail explanation of Spring Security and Spring Security OAuth, which is out of scope for this thesis project as this topic is more suitable for a book [71].

5.1 Libraries

Spring framework [72] is an open source solution for developing Java based enterprise applications. Previous implementation efforts in SWoT project have been based on Spring framework. In order to facilitate future integration of our prototype solution with the rest of the SWoT project we have

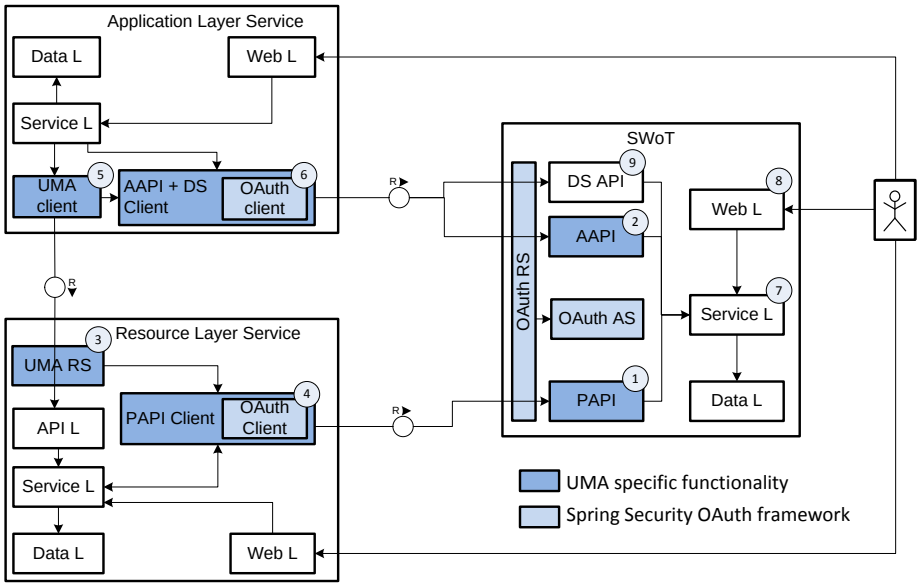


Figure 5.1: System implementation.

made a design decision to base our solution also on Spring framework. Spring framework is a mature solution with many extensions. One of those extensions is a Spring Security library [73], a comprehensive solution for addressing authentication and authorization requirements in Spring based applications. Its modular design enables development and integration of extensions with more specific functionality and support for more specific security mechanisms (such as different authentication methods and protocols [71]).

One of these Spring Security extensions is Spring Security OAuth library [74]. Spring Security OAuth features support for OAuth v1.0 and OAuth v2.0. It has been developed and maintained by Spring Source, creators of Spring framework. It is used in enterprise level commercial and open source products, such as Cloud Foundry [75] (open source platform as a service product).

In order to test and evaluate our UMA authorization server implementation which is part of SWoT framework we were required to implement UMA client and resource server roles even though this has not been our primary goal. We have implemented RU and RP applications also as Spring based solutions. This decision was motivated with the desire to reduce number of different solutions in use in order to reduce implementation effort and obviate

the need for learning additional solutions. We feel that choosing Spring for UMA authorization server implementation has been the correct decision. However, for RP applications which are deployed on resource constraint devices Spring based solution is inadequate. This is mainly due to high resource demands and significant codebase size of Spring based applications. Therefore, readers should note that RU and RP implementations do not fulfil all requirements present in production scenarios consider for SWoT framework.

One of the main contributions of this thesis project is Spring Security UMA (SSU) framework. SSU is a UMA framework written in Java, based on Spring Security and Spring Security OAuth. SSU provides functionality which is necessary to realize all UMA roles including a client, a resource server, and a authorization server.

We have designed SSU framework as a modular solution. SSU features three main modules which correspond to UMA roles. This modularity enables use of only those framework parts relevant for implementation of particular UMA role. For example, to realize RP application which acts as UMA resource server it is necessary to use only UMA resource server part of SSU framework.

Spring Security OAuth is a monolithic library that features support for OAuth roles (client, authorization server, and resource server) in a single tightly coupled codebase. However, to support modular design of SSU framework we needed parts of Spring Security OAuth. Therefore, we extracted parts which contained functionality required to realize OAuth client role and OAuth resource server role.

5.2 UMA Resource Server

UMA resource server component of SSU framework features two modules: first which fulfils function similar to function of OAuth resource server, and second which fulfils function similar to function of OAuth client.

The first module is responsible for receiving and processing resource requests originating from UMA clients. Larger part of this module's implementation originates from Spring Security OAuth library, more precisely from the part of the library which provides OAuth resource server functionality. This part handles integration with the Spring Security authentication and authorization mechanisms. For example, it provides implementations of important Spring Security artefacts, such as authentication processing filter, access denied handler, and access decision voter. In addition, it provides functionality for managing system internal objects that represent

authenticated subjects. Our contribution extends Spring Security OAuth resource server to address UMA specific requirements. This extension includes: functionality for handling UMA bearer tokens, functionality for generating and responding with UMA defined responses on UMA client requests, functionality for communicating parsed requesting party tokens to the second module for verification and permission retrieval, and functionality for making access decisions based on permissions associated with the requesting party token after it has been verified.

The second module is used to interact with the UMA authorization server as defined by UMA protocol. This module contains two submodules: first which contains OAuth client functionality, and second which contains UMA specific functionality. The first module is responsible for providing OAuth support to UMA resource server in its interaction with UMA authorization server. Larger part of this module is client functionality extracted from the Spring Security OAuth library. It provides functionality for managing OAuth client access tokens (such as functionality to retrieve, store, and embed tokens into HTTP messages) and for parsing and handling OAuth errors (such as expired access token). We have introduced relatively small set of modifications into this submodule and most of them with the intention to enable integration with the second submodule that contains UMA specific functionality. The second submodule relies on first submodule for OAuth support. The second submodule encapsulates client side functionality for interacting with the *protection API* of the UMA authorization server. This includes functionality for performing requesting party token verification and requesting party permission registration. In addition, it features support for processing UMA specific errors that may arise in communication with the UMA authorization server's *protection API*, such as invalid resource set id and invalid scope (see Section 3.2 in [60]).

During the implementation we have tried to follow UMA specification as close as possible. However, in several cases we diverged from UMA specification. For example, the design of Spring Security and OAuth libraries lead us to adopt slightly different approach than that suggested in UMA specification for performing token verification. In these solutions authentication and authorization are separated in time. First authentication is performed, during which all authorization attributes are retrieved from internal records (e.g. database record) and loaded into memory in form of an object representing authenticated subject. Then at some later moment in resource request processing authorization is performed. During authorization, authorization attributes from the object present in memory are compared with those required to access particular resource. Note that when authentication is performed there is no information about resource to

be accessed nor when authorization is performed retrieving of authorization attributes. This effectively prevents retrieval of authorization attributes for particular resource at the moment when authorization is performed (approach indicated by UMA specification). In the approach we have adopted UMA resource server in token verification interaction retrieves all permissions associated with the requesting party token in regard to all resources provided by the RP. Considering system performance this approach is more efficient as it reduces number of interactions between UMA resource and UMA authorization server. Due to change in response content in token verification interaction it was also necessary to modify corresponding format of the response.

We have designed UMA resource server part of SSU framework having in mind characteristics of our RP applications. These applications are relatively simple, single user applications, where user manages all application resources. Moreover, we have made a design decision to hardcode OAuth client credentials as their retrieval process was out of scope of this project. In addition, due to time constraints we were not able to implement some features which are necessary in realistic scenarios (e.g. caching of requesting party token information). Due to all these design decisions SSU framework in its current form is inadequate for more generic RP applications (multiuser RP applications) and more complex use cases (RP applications expected to interact with multiple SWoT instances simultaneously, or those that register with SWoT instances at runtime). For these use cases it will be necessary to extend SSU framework.

5.3 UMA Client

UMA client component of SSU framework features two modules: first which is similar to function of the barebone OAuth client (5), while second integrates OAuth client functionality to realize UMA client responsibilities in interaction with UMA authorization server (6).

First module is responsible for enabling communication between service layer of RU and entities acting as UMA resource servers. This module retrieves from internal records requesting party token and embeds it into corresponding HTTP requests, it parses responses indicating access denied and retrieves the ticket, and it forwards the ticket to the second module for following interactions with the UMA authorization server.

The second module integrates standard OAuth client functionality as all communication between UMA client and UMA authorization server is performed in particular user context, established by the OAuth protocol.

The functionality which builds on top of OAuth client module handles UMA specific interactions such as requesting party token issuing procedure, permission claim procedure (specified by UMA) and discovery of RP and resources (not specified by UMA). The UMA specified part is activated upon request from the first module, and upon finishing corresponding procedure adequate response is returned to the first module. The non UMA specific part which handles interaction with the DS API is activated upon request coming from the service layer of the RU. This part handles encoding and decoding of data present in messages communicated between the RU and the SWoT instance.

Current implementation of UMA client supports multiuser RU applications. However, as this is prototype implementation it has also certain limitations. For example, implementation supports establishing multiple UMA contexts per OAuth context, but only one OAuth context per user. Therefore, user using RU may interact with multiple RPs, but all RPs have to be part of same SWoT domain and to rely on same UMA authorization server for protection of resources which RPs host. There is no support for establishing initial trust (dynamic registration) between RU and SWoT instance. Instead, client credentials (client id and password) are hardcoded. Considering discovery service its implementation at this stage is quite rudimentary with basic support for discovering RPs rather than specific resources. RU applications are expected to feature built-in knowledge of semantics of APIs provided by discovered RPs.

While implementing RU application we have found similarity between how we handle RP discovery and communication with RP with how this functions have been performed in distributed Enterprise JavaBeans EJB based applications. More precisely, discovery of RPs is conceptually similar with the Java Naming and Directory Interface (JNDI) mechanism, while interaction between RU and RP is conceptually similar with the interaction between the client and the remote beans. Noticing this similarity in concepts on abstract level, enabled us to apply well established and popular design patterns for same problems. For example, we relied on service locator pattern for implementing discovery of RPs, and we have adopted business delegate pattern for abstracting communication with remote entities from inner parts of RU implementation.

5.4 UMA Authorization Server

SWoT features UMA authorization server functionality as its core part. This functionality includes protection API, directed towards UMA resource server,

and an authorization API, directed towards UMA clients. Both APIs are protected by the OAuth. OAuth resource server intercepts all requests directed towards these two APIs (see Figures 5.1, 5.2, 5.3), verifies them in interaction with OAuth authorization server (which in our case is tightly integrated with the OAuth resource server), and based on the verification result allows or denies further request processing.

Both protection and authorization API are implemented as three tier web applications in which functionality is divided into the web, service and data layer. For the implementation of the web layer we relied on Spring Model View Controller (MVC) framework. As our overall solution is Spring framework based the design decision to choose this particular MVC framework for the web layer was logical. In addition, in this implementation effort we have found beneficial that Spring MVC features strong support for building REST based web APIs. Main responsibility of the web layer as we have implemented it is to decode requests and encode (based on JavaScript Object Notation (JSON) format) results of processing performed in the service layer. The service layer contains core business logic: functionality to retrieve required internal records from the data layer, functionality that performs required checks, and functionality that processes request autonomously or in cooperation with XACML based service for specific requests (see Figure 5.3). For the implementation of the data access layer we relied on a persistence framework MyBatis [76]. MyBatis provides support for custom SQL queries, advanced mappings, and automatic generation of Java interfaces based on database schema. These MyBatis features enabled us to handle interaction with the database in elegant and seamless way. The last piece of the infrastructure in our solution is the MySQL database used for storing internal records including: OAuth artefacts, UMA artefacts, client, user credentials, and descriptions of resources provided by RPs.

Protection API consists of three separate functions which are implemented as three distinct web controllers (see Figure 5.2): resource registration controller, requesting party token status controller, and permission registration controller. Controllers receive requests, and verify that the content of the request matches that required for the function which they implement. If verification is successful controllers invoke corresponding functionality provided by resource registration service, requesting party token status service, and permission registration service. These three service layer components are required to interact with objects from data access layer to accomplish their function.

Resource registration service retrieves information from the resource provider data access object, and retrieves and stores information from/to resource and scope (actions applicable on the resource) data access objects

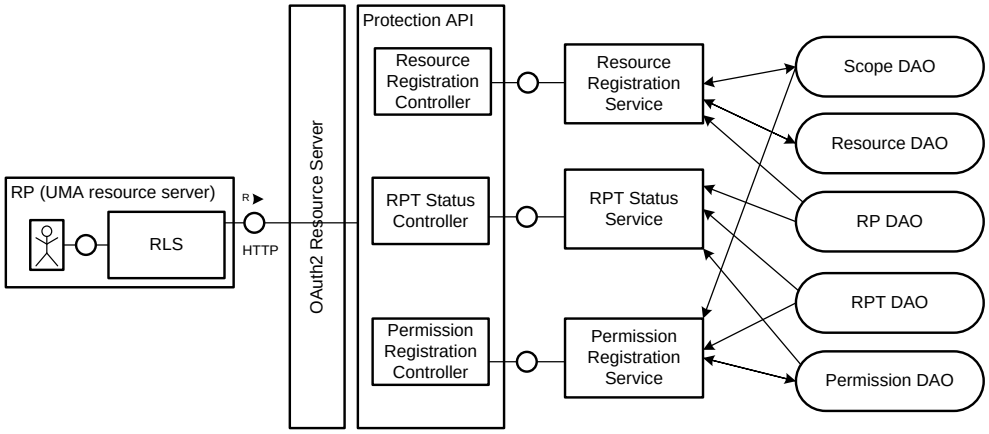


Figure 5.2: Protection API.

in its operation. Information from resource provider data access object is required for performing certain checks, as well as retrieving its identifier so that in operations such as storing resource description adequate internal record refers to correct RP which registers the resource. Scope and resource data objects contain information about scopes and resources respectively.

RPT status service retrieves information from resource provider, requesting party token, and permission data access objects in order to fulfil its function. Requesting party token provided in the request originating from the RP is used to locate the corresponding internal record in requesting party token data access object. If this is successful then RPT status service retrieves information about permissions associated with the particular requesting party token from permission data access object. The retrieved permissions are provided in response to the RP (UMA resource server) request. Information from the resource provider data access object is required to parametrize queries directed towards other two data access objects with which RPT status service interacts.

Permission registration service retrieves information from the scope, requesting party token, and permission data access object, while it stores information into the permission data access object. In request originating from the RP requesting party token and scopes to be assigned to requesting party token are provided. Since these artefacts are only handles to internal records permission registration service is required, by relying on these artefacts, to retrieve corresponding internal records from data access objects. When permission registration service retrieves data from scope and

requesting party token data access objects it generates internal record, which represents permission to be claimed identified by the new handle known as ticket, in permission data access object. The ticket itself is provided in response to the RP (UMA resource server).

Authorization API consists of two separate functions implemented by two different web controllers (see Figure 5.3): requesting party token request controller, and permission request controller. As in the case of protection API these authorization API controllers are responsible for decoding, encoding, and verifying contents of requests and responses. However, for real processing of requests web controllers invoke functionality in their associated service layer components: requesting party token request service, and permission request service. These two service components in their operation interact with several data access objects that abstract different database records.

Requesting party token request service stores and retrieves information to/from resource user and requesting party token data access object, and retrieves information from the resource provider data access object. When RU (UMA client) request requesting party token in its request it provides an identifier of the RP. Requesting party token request service retrieves internal resource provider record identified by the provided identifier, and internal resource user record. Based on the information present in these records RPT request service generates new requesting party token record which represents creation of the new UMA context. Each requesting party token record is identified by the requesting party token which is provided in the response to the RU.

Permission request service is a core component in the overall system. It is the place where new permissions are granted or denied. Permission request service in its operation retrieves information from several different data access objects including: resource user, requesting party token, resource provider, resource, and permission data access object. Permission request service also stores information in RPT data access object after permission have been successfully granted and removes records from the permission data access object after they have been claimed or expired. Since requesting party token is provided in the request coming from the RU (UMA client) internal record from RPT data access object is the first retrieved. Using information present in retrieved RPT record, permission request service retrieves other records such as resource provider record. The artefact which is also provided as part of UMA client request is the ticket. Permission service using ticket retrieves internal record from the permission data access object containing set of scopes that need to be granted to RU in order to successfully perform operation on RP's resources. Permission request service provides all this information (identity of the party accessing resources, resources which are

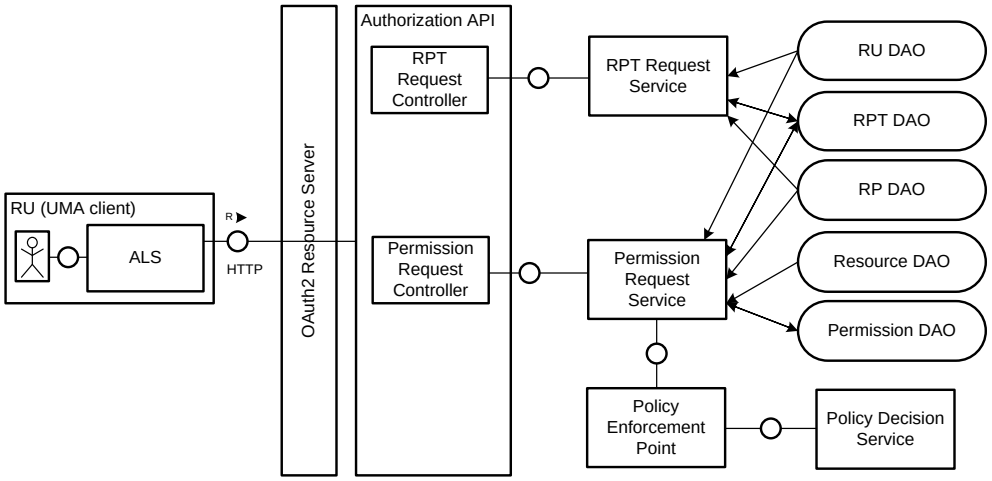


Figure 5.3: Authorization API.

accessed, and set of desired actions on the resources) to the XACML part of the system (policy enforcement point and policy decision point) for evaluation if requested scopes (actions) should be allowed or denied. If the response from the XACML part of the system is that these actions are allowed, permission request service adds these actions to the internal requesting party token record accompanied with their expiration time. If the response from the XACML part of the system is that these actions are not allowed, permission request service does not modify internal requesting party token record. In first case RU (UMA client) receives response indicating success, in second case RU receives response indicating that new privileges have not been granted to the RU and consequently that RU cannot access particular resource.

Chapter 6

Analysis and Discussion

In this chapter we present an analysis of our solution based on three different aspects: functional - how the system satisfies the functional requirements, security - how the system addresses the security requirements, and performance - how the system performs. This chapter ends with a discussion of the most interesting issues that we have considered during this thesis project.

6.1 Functional analysis

To evaluate our system we designed and implemented simple use case. We selected a use case in which a user controls numerous smart lights with the help of a single service. We were inspired to utilize this use case as it is used in the Lifx project [22]. To realize this use case we developed two applications: a light device application which represents a RLS and a light controller application which represents an ALS.

6.1.1 Use case components

The *Light Device Application (LDA)* is a simple, *single user* web application hosted at the device owner's premises. It provides access to the light device's functionality, e.g. checking if a light is on or off or to change its state, through two different interfaces. The first interface is an administration interface through which the device owner may directly interact with the light device. Using the functionality provided via this first interface the owner manages (activates, deactivates) a second LDA interface which is light service API. This light service API is offered to other applications so that they can access the light device's functionality. Applications using the light's service API are

developed with built in knowledge of this API's semantics. The light service API offers a single resource with two operations:

- GET /light - Check light state
- POST /light - Change light state

As unlimited access to these resources is undesirable, appropriate access control mechanisms need to be in place to protect and limit access to resources of this type. It would be inconvenient for the user to manage access control directly at the LDA. This is why the LDA supports the UMA protocol, more precisely the LDA acts as UMA RS. The LDA enables a user to request externalization of resource protection to the SWoT platform (configured as a UMA AS at development time). After a resource has been registered with the SWoT in response to particular user action, the LDA starts processing resource requests at the light service API in accordance with the UMA protocol.

The *Light Controller Application (LCA)* is a *multi user* web application. It is a mashup application which combines instances of the same service API offered by different resource providers, e.g. LDA. In this it differs from current mashup applications which typically combine different service APIs. LCA can interact with and control light devices which implement and expose the light service API. Using LCA users invoke operations targeting a specific light device or group of light devices. For example, users of LCA could preconfigure which lights should be turned on or turned off in their home for a particular setting, such as party or movie playback and then later easily invoke this specific light configuration. The LCA could be extended to enable users to configure lights to turn on or off at specific time by utilizing time attributes.

LCA was developed to follow the UMA protocol, more precisely LCA acts as UMA client. After a user authorizes the LDA to interact with the SWoT, the LCA finds all of the light devices that are discoverable via the SWoT's DS in the user's context. The LCA uses the retrieved information to present all of these instances via a GUI. The user through the GUI requests operations on particular light devices and the LCA interacts via the light service API of a specific LDA to perform this operation. The result of the operation is returned to the user via the GUI. In order to perform operations on LDA resources the LCA needs adequate permissions. The SWoT's UMA AS grants these permissions based on the registered resource's access policies defined by the user.

6.1.2 Evaluation test cases

Fir our test cases we assume that a user has several smart lights. Each light has its own instance of LDA. Initially the user accesses each LDA application and requests the resources to be protected by the user's SWoT framework instance. After this the user is no longer required to interact directly with any LDA.

The user interacts indirectly with LDAs indirectly by utilizing a LCA. However, before this can happen the user needs to authorize the LCA to interact with their SWoT. As a result of this authorization this LCA becomes a member of the user's SWoT domain and it is allowed to access the resource discovery service.

The user of the SWoT framework can view all LDAs that are part of his or her SWoT domain, as well as all of the resources that they have registered. The user can also view all the ALSs that are part of his or her domain. In this concrete use case there is only one - LCA. Via the user's SWoT the user is able to define attributes, attribute values, and assign these attributes to the resources or to the applications (such as the LCA).

Test case 1: The default SWoT resource access policy is to deny resource access. Therefore if a user has not specified any access policy or none of the available policies is applicable then the LCA will not be able to obtain permission and consequently access the resource.

Precondition: User has not defined any access policy.
Success scenario: User initiates action at LCA to check the state or change the state of a specific LDA. The action results in a error report being displayed at the LCA informing the user that this LCA is not authorized for this action and that the user should modify policies which apply to the LCA at the user's SWoT instance.
Failure scenario: The user initiates an action at the LCA to check the state or change the state of a specific LDA. Action is performed successfully.

Test case 2: The user defines an attribute *trust level* and range of attribute values applicable to it: *low*, *medium*, and *high*. Next it assigns a low trust level to the LCA. Then it creates a XACML access policy which grants all applications (subjects in XACML terminology) with low trust level access to check any LDA's state resource.

Success scenario: The user initiates action at the LCA to check the state of a specific LDA. The action is successful.

Failure scenario: The user initiates action at the LCA to check the state of a specific LDA. However, action returns an error indicating that the LCA is not authorized.

Test case 3: The user sets the LCA's trust level to medium. Then the user creates a new XACML access policy which grants all applications with medium trust level, access to any check or change the state of any LDA.

Success scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. The action is successful.
Failure scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. However, action returns an error indicating that LCA is not authorized. The action is unsuccessful.

Test case 4: User modifies a XACML access policy to add a time based constraint, such that the access policy allows all applications with a medium trust level to access and check the state or change the state of any LDA's resource, but only during the period from 17:00 to 23:00 (e.g. when this user is usually home).

Precondition: current time outside of constrained time period.
Success scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. Action results in an error report displayed at the LCA informing the user that the LCA is not authorized for this action and that the user should modify policies which apply to the LCA at the user's SWoT instance.
Failure scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. Action is successfully performed.

Precondition: current time is part of constrained time period.
Success scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. Action is successfully performed.
Failure scenario: The user initiates action at the LCA to check the state or change the state of a specific LDA. Action results in an error report displayed at the LCA informing the user that the LCA is not authorized for this action and that the user should modify policies which apply to the LCA at the user's SWoT instance.

6.2 Security analysis

Our overall system is a set of technical measures for addressing threats according to the high level threat model presented in Chapter 2. However, on a more detailed level OAuth and UMA entities may also be at risk of a compromise. A comprehensive threat model for the OAuth 2.0 protocol discussing security features built into the protocol, threats, countermeasures, and assumptions under which they apply is given in [77]. However, comprehensive threat model for UMA has not yet been developed. The UMA specification refers to the OAuth threat model [77] for security considerations. Although contributions from OAuth's threat model are useful for UMA, we consider the OAuth threat model incomplete for UMA purposes as it covers only OAuth. However, UMA addresses more complex use case, hence it has artifacts (e.g. requesting party token) and interactions (with protection and authorization API) that OAuth does not have. We expect that there are additional threats due to the additional features.

Defining a comprehensive UMA threat model is out of the scope of this thesis project. Instead we provide a basic threat model covering a set of UMA specific threats and issues. In contrast to our high level threat model (described in Chapter 2) where the focus was on malicious internal parties, here we concentrate on outside attackers. Therefore, in our analysis we assume that the authorization server, resource server, and client correctly follow the UMA protocol and comply with all applicable obligations from the UMA binding obligations specification [25].

6.2.1 Overview of assumptions and features

In our analysis we assume that the attacker has full access to the network between all UMA parties: client - authorization server, client - resource server, and resource server - authorization server. As a result the attacker is able to eavesdrop on any communication between these parties. Moreover, the attacker has unlimited resources at their disposal for performing an attack.

Considering the parties involved in a system's operation we assume that the authorization server stores the following artifacts: user credentials, client credentials, OAuth credentials (refresh and access token), HTTPS certificate, and UMA specific artifacts (such as requesting party tokens and tickets). We also assume that each resource server stores: user data, HTTPS certificate, this resource server's credentials for each authorization server (i.e., its client ID and secret), OAuth credentials for each end user (i.e., a refresh and

protection access token), and UMA artifacts for each request (i.e., the requesting party token and ticket). The resource server does not have access to clients' secrets, clients' refresh tokens, or clients' authorization access token. Moreover, we expect that each client stores: its credentials for each authorization server (i.e., its client ID and secret), OAuth credentials for each end user (i.e., a refresh and authorization access token), and UMA credential for each resource server (i.e., a requesting party token).

UMA builds on top of OAuth by adding additional security features required to mitigate certain types of attacks and security issues. These additions include artifacts such as a requesting party token and a ticket. Both of these artifacts in the current UMA specification are primarily described by usage of a handle based token design. The UMA specification indicates that a requesting party token can also be implemented as an assertion (self contained token). Using an assertion design eliminates the need ti perform token validation at the resource server via an interaction with an authorization server. However, due to the UMA protocol design using an assertion design for a ticket is inadequate as this approach does not bring any benefits. Detailed explanation of handles and assertions is given in Section 3.1 of [77]).

The requesting party token can be used in requests to resource servers as a bearer token or proof token. A bearer token is more convenient for client applications as its possession is all that is needed for its usage. In contrast, a proof token is less convenient, but more secure as it is digitally signed with a secret corresponding to that particular token.

OAuth defines scope and limited lifetime as token constraints resulting in an increase in the overall system security. Considering UMA specific artifacts (i.e., a requesting party token and a ticket), we recognize that a limited lifetime is an applicable constraint. However, scope constraint is unnecessary for the requesting party token and ticket, since by UMA's protocol design these artifacts are already constrained to being used at a specific resource server and under specific conditions.

6.2.2 UMA threats

In this section we present an extension of the OAuth threat model with UMA specific threats. We first present threats related to the UMA specific artifact: requesting party token. We do not consider threats related to the ticket as they are identical to those related to requesting party token. In remainder of this section we discuss threats directed towards the UMA client - UMA resource server interaction. Where applicable we provide references to similar threats defined in OAuth threat model as described in [77].

Threat 1: Obtaining a requesting party token

The attacker can obtain a requesting party token directly from UMA parties having access to it. For example, an attacker may perform a SQL injection attacks, circumvent inadequate storage protection measures, or exploit misconfiguration of database security settings. When then requesting party token is a bearer token (such as in our solution), a successful attacker can access all resources to extent granted to the client and associated with the requesting party token. Countermeasures include mechanisms envisaged in the UMA protocol specification, such as using a proof token instead of a bearer token and more specific measures recommend by the OAuth threat model (see Section 4.1.3 in [77]):

- Keep a requesting party token only in transit memory (applicable to client and resource server),
- Store requesting party token hashes only (applicable to the authorization server),
- Keep the requesting party token lifetime short,
- Enforce system security measures,
- Enforce standard SQL injection countermeasures.

Threat 2: Eavesdropping requesting party token

In a handle based design, which we have adopted for our solution, a requesting party token is transported from authorization server to client, from client to resource server, and from resource server to authorization server as an artifact in different interactions. Therefore, there are multiple interception points where attackers may eavesdrop this communication. The impact of this threat is same as of *threat 1* in which attackers are also in possession of a requesting party token. However, here the countermeasures differ:

- Use of transport layer security mechanisms such as TLS [78, 79] to protect the communication between all parties (as recommended by the UMA specification) and
- If end-to-end confidentiality cannot be guaranteed, the extent of damages may be reduced by limiting permissions that can be associated with the token, by shortening the token's lifetime and/or permissions expiry time.

Threat 3: Guessing requesting party tokens (see Section 4.6.3 of [77])

If an attacker knows how the requesting party tokens are generated the attacker might try to guess them. If an attacker succeeds, then the attacker may have access to resources of a particular resource server to extent which a requesting party token that was guessed has. Countermeasures for this threat include:

- Requesting party tokens should have a high level of entropy, so that guessing valid token is infeasible,
- Usage of proof tokens thwarts this attack, as even though an attacker correctly guesses a token's value it cannot access protected resources, and
- Limiting a token lifetime may reduce possible damage in the absence of the previous more suitable measures.

Threat 4: Replay of authorized resource server requests (see Section 4.6.2 of [77])

An attacker may intercept and then later replay valid requests sent by a client to a resource server. This threat differs from others, as attacker does not need to possess confidential information, but still may succeed in achieving an unauthorized resource server operation. For example, if a resource server hosts functionality for locking and unlocking an entrance door, then an attacker by replaying previous valid requests from an authorized client for locking and unlocking door may achieve the same result even though the attacker is unauthorized. Countermeasures are:

- Usage of transport layer mechanisms, such as TLS, to prevent an attacker from capturing requests and
- Usage of additional security features, such as signed requests, proof tokens, nonces, and timestamps, to ensure that requests are fresh and unique.

Threat 5: Requesting party token phishing by counterfeit resource server (see Section 4.6.4 of [77])

An attacker may pretend to be a resource server. If the client provides its requesting party token to a counterfeit resource server, then the attacker

will be able to use this requesting party token to access resources at the resource server. An attack in which an attacker tunnels traffic from one party to another (the client and original resource server in our case) and back is known as man in the middle. Countermeasures are:

- The client should authenticate the resource server before presenting a requesting party token and
- Limiting token lifetime may reduce the damage in absence of more suitable measures.

Threat 6: Leak of confidential data (see Section 4.6.6 and 4.6.7 of [77])

Often requests on their way from client to resource server pass through HTTP proxies and caches. Moreover, these requests are often logged. Improper configuration of these mechanisms or weaknesses in the log's protection may enable attacker to access confidential artifacts, such as a requesting party token. Countermeasures include:

- Proper usage of HTTP mechanisms such, as Cache-Control to avoid caching of confidential artefacts,
- Proper configuration of logging mechanism,
- Restricting access to log files,
- Usage of proof tokens, and
- Limiting token lifetime to reduce possible damage.

Threat 7: Denial of service (DoS) attacks

An attacker who owns a botnet and has knowledge of the IP addresses of resource servers can request access providing random requesting party tokens. This results in establishing HTTPS sessions to each resource server which ties up some amount of resources. Furthermore, each resource server needs to validate the requesting party token with the central authorization service. As this interaction also requires establishing a HTTPS session it uses additional resources both at the resource server and authorization server. This may result in a DoS attack on a resource server which has limited impact or on an authorization server which has a high impact as the authorization server represents single point of failure. Countermeasure is:

- Usage of proof tokens or assertion based tokens to avoid tying up additional resources during token validation operation. The disadvantage of these measures is that system management becomes harder, e.g. distributing secrets, certificates, etc.

6.3 Performance analysis

In this section we discuss major factors affecting our system's performance including: UMA interactions, permission expiration period, resource granularity, and application interaction patterns. We also provide results of performance measurements for our simple use case. Our future work will include comprehensive performance analysis for a more complex application scenario with concrete deployment scenario.

6.3.1 UMA flows

The UMA protocol is an authorization protocol that features three different protocol flows: *obtaining RPT flow* establishes the authorization context, through *first resource access flow* client retrieves missing permissions, and in *second and subsequent resource access flow* client accesses the resource with all necessary permissions. These flows (illustrated in Figure 6.1) consist of different number of message exchanges. In addition, the frequency by which these flows are repeated within the system varies. For example, *obtaining RPT flow* is performed whenever an authorization context has to be established which happens relatively rare compared to other two flows. *First resource access flow* is performed whenever client tries to access resource without required permissions. *Second and subsequent resource access flow* is performed whenever client has all necessary permissions to access resource and in general it is the flow most frequently repeated.

6.3.1.1 Obtaining RPT flow

This flow of the protocol includes two request/response exchanges (the first of which is optional). The first interaction is between a client and a RS (node CR in Figure 6.1) and results in an access denied response. From this interaction the client learns where the authoritative AS for this particular RS is located. This interaction is required in systems where a client does not have an already established relation with the AS. In our system, all third party entities are preconfigured to interact with specific SWoT instance. Therefore, this interaction is optional. In the second interaction the client obtains from

Table 6.1: Request/Response exchanges in UMA Protocol.

Symbol	Description	Client	Server
CR	Resource Access	Client	RS
RA1	Retrieve RPT meta-information	RS	AS
RA2	Register client permission	RS	AS
CA1	Retrieve RPT	Client	AS
CA2	Claim permission	Client	AS

the AS a RPT for a specific RS by providing the RS identifier (node CA1 in Figure 6.1). In our system the client obtains this RS identifier by interacting with the SWoT's DS, in systems that strictly follow UMA protocol the client receives this identifier in an access denied response from the RS.

6.3.1.2 First resource access flow

This flow requires at least six request/response exchanges assuming following:

- **A1:** A client provides a valid RPT in its interaction with the RS.
- **A2:** The RPT that a client provides to the RS does not have any permissions associated with it that are applicable for resource being accessed.

Client initiates the process by requesting resource access at the RS (CR in Figure 6.1). The RS in interaction with the AS retrieves the RPT meta-information and permissions associated with it (RA1 in Figure 6.1). Since this RPT is valid (by assumption A1) and does not have the necessary permissions (by assumption A2) the RS registers the permission in an interaction with AS (RA2 in Figure 6.1). The result of this interaction is a ticket which the RS provides to a client as a response in the previous CR exchange. This process continues with an interaction between the client and AS in which the client claims permissions by providing a ticket and a RPT (CA2 in Figure 6.1). Two outcomes are possible as a result of this CA2 interaction:

- Permission association is not allowed. The AS response indicates a failure and therefore the client cannot access this resource.
- Permission association is allowed. The AS internally associates this permission with the RPT and returns a response indicating success.

In case of a successful CA2 interaction, then the client tries again to access the resource at the RS (CR in Figure 6.1). The RS in interaction with the

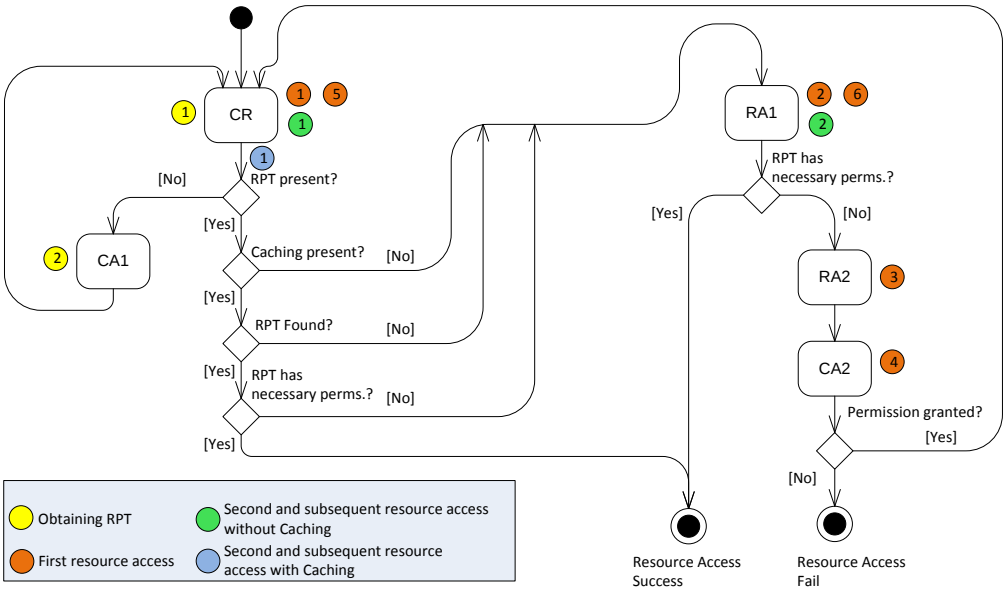


Figure 6.1: Request/Response exchanges in UMA protocol. Rectangles indicate different interactions (their definition is available in Table 6.1, numbers enumerate their order in different cases marked by different colors).

AS retrieves the RPT's status and permissions associated with it (RA1 in Figure 6.1). Since this RPT is valid (by assumption A1) and now it has the necessary permissions the RS satisfies the resource access request. In summary six interactions were necessary to successfully access a resource for the first time (CR, RA1, RA2, CA2, CR, RA1 in Figure 6.1).

6.3.1.3 Second and subsequent resource access flow

This flow requires two request/response exchanges when the RS does not utilize caching. Otherwise, only one request/response exchange is required. This case assumes the following:

- **A3:** Previous resource access was successful.
- **A4:** Permissions previously associated with the RPT that were necessary for accessing the resource have not been invalidated in the meantime.

The client as in the previous case initiates the process by requesting resource access at the RS (CR). The RS in interaction with the AS retrieves the RPT's

meta-information and permissions associated with it (RA1 in Figure 6.1). Since this RPT is valid (by the assumption A1) and it has the necessary permissions (by assumptions A3 and A4) the RS satisfies the resource access request. If assumption A4 does not hold, then the process continues with interactions as in the first resource access starting with the RA2 interaction. This case can be optimized by caching RPT's meta-information at the RS. If this optimization is applied, then only one interaction is required (CR).

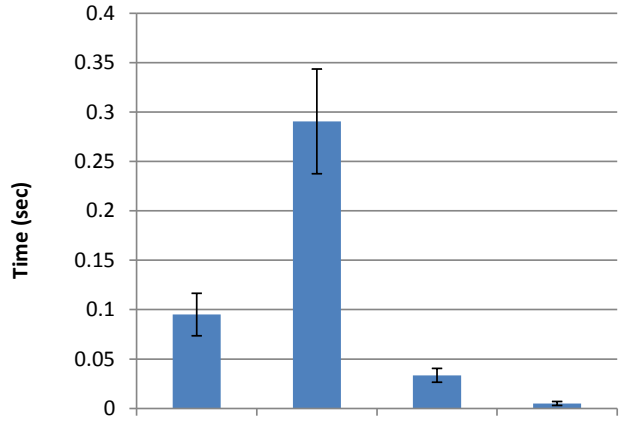
6.3.2 Performance of UMA flows

Performance evaluation of distributed systems is difficult since it has to consider the *computational performance* of all components involved in the system's operation, as well as the *communication overhead* present in the system's (normal) operation [80]. In Internet based distributed systems, it is common that communication overhead has a much larger impact on the overall system's performance than the computational performance. However, measuring communication overhead is hard as it is affected by many non controllable factors (such as, network throughput, network congestion, physical distance, etc.).

In our performance evaluation setup all three system entities are running on the same machine. LDA, LCA, and SWoT framework instance are executing in a virtual machine running Ubuntu 11.04 Natty Narwhal, 32-bit operating system, with 2GB RAM, and a 2.2GHz Intel Core 2 Duo processor. While this setup does not provide realistic overall performance measurements (collocating system entities minimizes communication overhead), it reduces the complexity of evaluating computation performance. This is mainly because homogeneous execution environment establishes common referent system (there are no variations due to heterogeneous hardware resources) and timing measurements are synchronized (they rely on a same clock).

We have utilized application level logs as a source of timing information. We recorded 20 observations for each flow (described in subsection 6.3.1). In addition to recording the time necessary to perform whole flow (our main variable of interest) we have recorded timing information for each interaction (such as, retrieving RPT token, retrieving RPT status, registering permission, etc.). Moreover, we have instrumentalized application code in a particular way so that we can differentiate between the time which system entities spend in processing inputs from the time which system entities spend in communication. Therefore, our measurements allow us to assess overall system performance both on a coarse and fine grain level.

From obtained measurements we have determined that the time necessary to perform each flow cannot be characterized by the normal distribution. In-



	Obtaining RPT	1st Resource Access	2nd Resource Access	2nd Res. Access with Caching
Median:	0.095	0.2905	0.0335	0.005
MAD:	0.0215	0.053	0.007	0.002

Figure 6.2: Time measurements for different UMA flows.

stead, the measurements resemble more complex long tail distributions (such as, log-normal, Weibull, or Rayleigh distribution). All these distributions as well as our measurements feature positive skew where the mass of the distribution is on the left side and a long tail of outliers on the right side of the graph. Moreover, due to characteristics of these distributions the mean and the standard deviation are inappropriate statistics as they are strongly affected by the outliers. On the other hand, the median and the median absolute deviation are more appropriate statistics as they are resistant to outliers. Therefore, we use the median and the median absolute deviation to characterize the time necessary to perform different flows (illustrated in Figure 6.2).

Even though retrieving RPT token is a single interaction flow, similar to second resource access flow with caching applied, it takes more time than the second resource access flow in both modes (with and without caching). This is mainly due to the complexity of retrieving RPT token operation, which at the UMA authorization server includes multiple database operations (both inserts and queries). In contrast, second resource access interactions (such as, retrieving RPT status and resource access) often do not interact with the database due to the application of caching mechanisms or have small number of database read operations. We feel that there is a lot of place for

improvement in our database design which backs UMA authorization server. These improvements when realized may result in a more optimized database operations and more efficient system interactions which rely heavily on the database (such as retrieving RPT token).

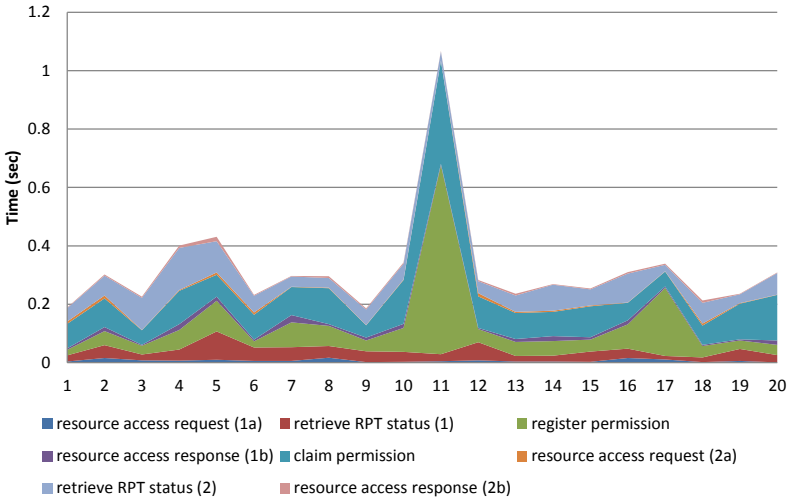
As shown in the graph (Figure 6.2), the first resource access flow takes significantly more time than the other flows. This does not come as a surprise as the first resource access flow is the most complex flow which embeds multiple interactions. The interactions which are part of first resource flow have been described in subsection 6.3.1 and there individual contribution to the duration of the overall flow is represented in Figure 6.3a. The interactions which dominate the first resource access flow are permission registration and permission claim operation. Figure 6.3a presents these two operations as integral units even though they consist of components which describe processing time, communicating request, and communicating response times. In Figure 6.3b and Figure 6.3c we depict these three components separately for permission registration and permission claim operation respectively. It is noticeable from these graphs that the time for performing these two operations mainly depends on the time UMA authorization server takes for performing them.

Second resource access flow measurements are present in the graph in two forms (with and without caching). The results indicate that with the caching, second resource access flow takes approximately seven times less time than without the caching applied. Overall second resource access flow is the shortest operation with the smallest computation and communication overhead.

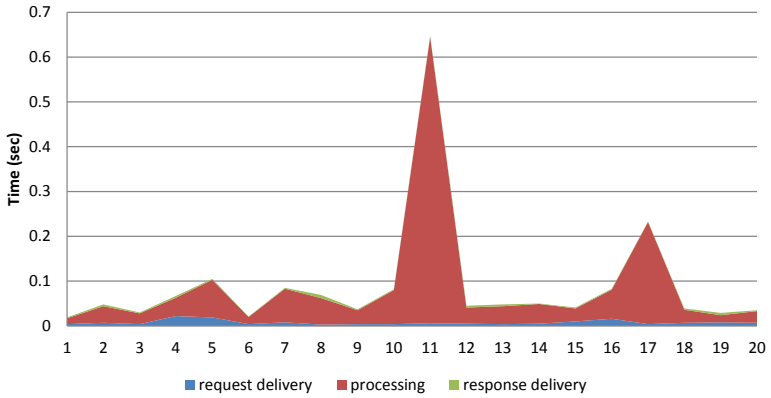
We have performed measurements for both GET and POST based resource access operation. Our results show that in all three flows which contain resource access interaction (first resource access, second resource access without caching, and second resource access with caching) the GET operation is slightly faster. However, the difference in duration between same flows with GET and POST operation is small and can be attributed to the overhead in encoding and decoding parameters present during POST operation. As these observations do not provide any relevant insights for this thesis project we have left out discussion and measurements for flows with POST based resource access interaction.

6.3.3 Parameters affecting performance

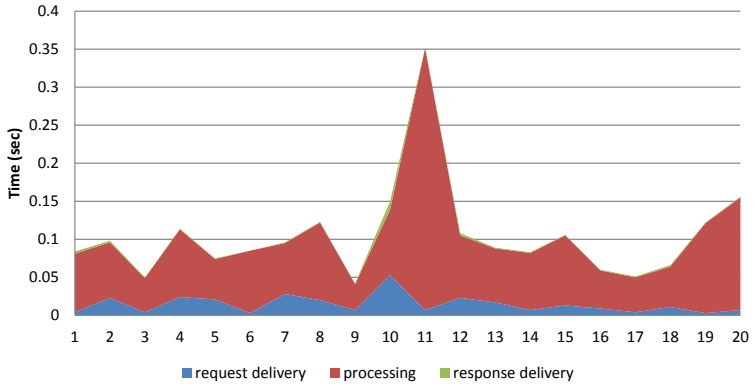
In order to increase the performance of the overall system it is necessary to minimize the number of non-functional interactions within the system. This means that the frequency of repeating certain UMA flows should be big



(a) All operations combined.



(b) Permission registration operation.



(c) Permission claim operation.

Figure 6.3: First resource access flow measurements.

enough to support system's security requirements but still small enough to not introduce undesirable level of overhead. As the security and performance requirements vary between different application use case scenarios, we cannot determine exact value appropriate in all use case scenarios. Instead, we can identify parameters which affect the frequency of UMA flows. These parameters will be utilized for modelling system performance in different applications, planned for future work.

6.3.3.1 Permission expiration period

Permissions associated with the RPT expire. When permissions expire the client has to go through the whole process of obtaining permission which results in significant overhead. How often this happens depends on the permission expiration period value. By modifying the value of this parameter we can alternate the system's behavior. For example, if permission expiration period is small enough, but not smaller than the time a client needs to access a resource at least once, then permissions expire often and the system exhibits dynamic access control. A side effect of this approach is the significant communication overhead due to required UMA interactions. If permission expiration period is set to have a large value than permissions rarely expire, and as they are cached at the RS communication overhead is reduced to a minimum. This approach also requires the introduction of a permission revocation mechanism. UMA currently does not feature permission revocation mechanism.

For our implementation we have adopted an approach where all permissions get same permission expiration period, as this approach is the simplest. However, we envisage an alternative approach in which different permission expiration period values are assigned to different permissions. For example, for operation on a sensitive resource, such as opening door, the permission expiration period value assigned may be small so that it is necessary to go through permission obtaining procedure more often, but for other less important resources such as manipulating a light a larger permission expiration period value may be assigned. However, this approach requires deeper understanding of resources, clients, and their application interactions. Therefore, the approach for handling permission expiration period parameter depends on the specific system requirements.

6.3.3.2 Resource granularity

Another factor that has impact on the system's performance is the organization of resources by a RS. When a RS manages resources on fine

grained level, e.g every sensor is a different resource, this enables a fine grained access decision making process. For example, it is possible to allow a particular client to access one sensor but not any other. However, this affects the system's performance. For example, client needs to go through UMA procedure for obtaining permission twice in order to access two different sensors. If a RS registered all sensors as a single resource, then a resource owner would be able to grant only coarse grained permission. However, a client which obtains this permission would be able to access all sensors and would not be required to go through multiple UMA permission obtaining procedures.

Alternative approach to handle resource organization is using resource hierarchies. This approach may result in better system performance as it can reduce number of UMA interactions. For example, client can be granted permission to access parent resource which will allow it to access all its child resources at the same time. Resources hierarchies may be defined by RS or based on standardized classifications (ontologies) which could facilitate their management and implementation in authorization server. Resource hierarchies in some extent resemble hierarchies in file systems. Therefore, there is possibility that approaches for handling permissions in file systems may be applied for handling permissions in resource hierarchies. UMA currently does not define approach for handling resource hierarchies.

6.3.3.3 Application interaction patterns

Application interaction patterns may also affect system performance. For example, an application which interacts with a single resource provider differs from a mashup application which interacts with multiple resource providers even though their function may look alike to the user. The most relevant difference for our analysis, is that mashup application must establish a security context for each resource provider independently. As security related interactions negatively affect performance the user may experience large variations in perceived performance between applications with different interaction patterns due to these underlying causes.

Within SENSEI project the problem of establishing multiple security contexts by single application has been identified and explored. Moreover, special component in the SENSEI security architecture, named resource access proxy service, has been introduced which can establish security context for multiple resource providers on behalf of client applications [12]. This approach results in a more efficient system operation for more complex application interaction patterns. However, it also makes overall system more complex as implementation of different application interaction patterns

within the system differs.

6.4 Discussion

The goal of this thesis project was to design and prototype an authorization solution for a user centric integration of application and resource layer services envisaged in a future Web enhanced with smart things. We discovered requirements and defined an authorization architecture by studying various platforms, frameworks, and their approaches for service integration. However, SENSEI's project contributions had the largest impact on this aspect of our work.

After defining the authorization architecture we studied how this architecture could be realized. We analysed various security mechanisms considering their characteristics and how they relate to each other. Based on our analysis of selected mechanisms we determined that our problem requires application of two protocols UMA and OAuth. We also specified how these mechanisms should be incorporated into SWoT framework. This specification included discussion of how services join SWoT domain, how users administer resources' access policies, and how the permission requests are evaluated.

Even with a working design, the implementation phase required significant effort. The majority of our implementation effort went into implementing the UMA framework, as at the time there was no publicly available implementation. In addition, Spring Security and Spring Security OAuth libraries, which our solution integrates and adapts, represent relatively complex solutions to learn. However, we believe that basing our work on these two enterprise grade libraries has increased the quality of our implementation.

Our functional evaluation shows that our solution facilitates administration of multiple resource providers and that its utility increases as the number of resource providers increases. The solution also simplifies implementation of the resource providers as it provides pluggable module for addressing their security requirements. Our design decision to extract evaluation of permission requests into a special module, ensures that the solution can be enhanced with more flexible mechanisms in the future. For example, the solution may easily be modified to support dynamic access control by replacing our evaluation solution with XACML.

Our security evaluation and the analysis of countermeasures for threats related to the OAuth (given in [77]) and UMA (discussed in Section 6.2) indicates that the usage of transport layer security mechanism, such as TLS, is required for interactions between all parties in UMA system. Consequently, SWoT entities should adopt TLS, or an adequate alternative mechanism, to

protect the communication. In addition, assertion based token design, even though more complex than handle based token design, increases the security of the overall system and may improve system performance by eliminating the need for token validation interactions.

Our current system implementation does *not* use transport layer security mechanism nor an assertion based design for tokens. Our limited time and resources for this thesis project are the primary reasons for not adopting these mechanisms. Moreover, our high level approach to the problem without details of the particular deployment scenario prevented us from making concrete design decisions. For example, for resource servers implemented on resource constrained devices the TLS protocol may not be the proper solution and additional analysis is required [68][81]. Therefore, implementation improvements which address the above recommendations and take into consideration the requirements of a particular deployment scenario will be part of future work.

Our performance evaluation in Section 6.3 indicates that on initial resource access the overhead of UMA related interactions is significant. However, subsequent requests to same resource do not introduce significant overhead, especially when the resource server caches the RPT's meta-information. Our analysis identified that the permission expiration period, resource granularity, and application interaction patterns may have significant impact on the overall system's performance. Since these parameters are application dependent their analysis is infeasible without more realistic application use cases. Therefore, detail analysis of previously mentioned parameters and of their impact on the system performance is left for future work.

Chapter 7

Conclusions

This final chapter provides a short summary of the thesis contributions and describes directions for future work. Moreover, we provide summary of our reflections on social, economic, and ethical aspects of this thesis.

7.1 Summary

The goal of this thesis project was to explore authorization requirements and architecture for user controllable integration of smart things and services. We started our research by comparing contemporary solutions from different domains. Results of this study enabled us to identify role model solutions on which to base our work. Contributions from the SENSEI project enabled us to define suitable authorization architecture. Contributions from the Cloud OS concept solution enabled us to recognize specific requirements which novel user centric solutions introduce.

To realize desired authorization architecture it was necessary to identify suitable authorization mechanisms. Therefore, we performed extensive study of various security protocols considering previously identified authorization requirements. We determined that UMA protocol which integrates OAuth protocol is adequate mechanism for the envisaged authorization architecture. We proceeded with the specification of the UMA application in SWoT.

In addition, we have implemented designed authorization architecture as a prototype solution. Our UMA framework implementation represents its core part. It incorporates Spring Security and Spring Security OAuth libraries which provide necessary underlying functionality. Our implementation effort also included development of software agents prototypes which could be deployed on smart things or in the cloud, for the purpose of evaluating proposed authorization solution.

We evaluated our prototype solution in extent feasible for its capabilities. This study included functional evaluation, where we analysed if our implementation follows the specification, security evaluation, where we explored what are the threats and their possible countermeasures, and performance evaluation, where we analysed how much overhead do security related interactions introduce into the system operation.

In this thesis we have explored subset of SWoT's security requirements. As a result, our suggested solution represents only a small piece of future more comprehensive security solution that has yet to be developed. However, we strongly believe that our work performed in this thesis provides a solid foundation for this future effort.

7.2 Future Work

Research performed during this thesis project has revealed several alternative directions for future work.

First direction encompasses efforts which focus on characteristics and requirements of a smart thing rather than focusing on platform as in this thesis project. Majority of research performed within IoT research domain takes this perspective. As a result there is a vast body of related work that will need to be analysed and incorporated. Our efforts along this direction will analyse applicability of our authorization framework for smart things, taking into account their resource and deployment environment limitations. We expect that analysis results will elicit more specific security requirements than those considered in this thesis project. Consequently, authorization and corresponding evaluation framework will be improved.

Second direction features efforts which will extend proposed authorization framework and transform it into a comprehensive security solution. As a first step in this direction we suggest evaluating integration with XACML. This access control mechanism enables flexible system administration based on fine grained and dynamically evaluated access control policies. However, XACML is a complex solution designed for expert level users. Therefore, previous effort will need to be coupled with studies and contributions from usability domain. Following steps may include integration and evaluation of suitable authentication and accounting mechanisms.

Third direction encapsulates studies which will analyse requirements and propose solutions for realization of more flexible use case scenarios. In use case scenario considered in this thesis project all software agents act on behalf of same human user. In future work we will consider more generic use case scenario featuring interaction between software agents operating on behalf

of different entities (both natural and legal persons). We envisage that this use case will put emphasis on authentication system and introduce the need for incorporating additional UMA features (such as UMA Claims). Another option along this same direction would be to explore possible solutions for federating SWoT instances. Similar use case scenario has been explored within SENSEI project, but on a more abstract level.

7.3 Reflections

Today, it is common business practice for web service providers to generate profit by monetizing user generated data. In return, users are entitled to use service without paying for it. However, users rarely receive any kind of monetary compensation. Moreover, in this exchange they harm their own privacy by sharing their personal information with service providers. These practices in which real value of information is concealed for the benefit of service providers may have far reaching implications on the society [82]. Same business practices with small variations can also be applied in future IoT. For example, smart things may be provided for free in exchange for exclusive access to information which they generate. If these practices become widely adopted we may witness further erosion of users privacy as well as continuation of redistribution of wealth towards service providers. Alternative approach envisages enabling users to monetize their data and by extension data produced by their smart things [83][84][82]. Our solution architecture is amenable for such mode of operation. However, it needs to be extended with suitable accounting and billing mechanisms.

Our solution enables smart thing owners to mediate communication between smart things and services. As a result of decoupling smart things and services, economic incentives emerge which are more aligned with the interests of smart things owners. For example, device manufacturers may compete based on the price, features, and interoperability of smart things with different services. Service providers may compete based on the price, features, and their adoption of socially acceptable practices. Moreover, architecture design minimizes effects of data lock in practices. For example, when user decides to replace particular service there are no costs, nor waste, as the smart things and their capabilities may be reused by other services. Therefore, service providers have incentive to respect user's privacy as otherwise they could be replaced.

Different legal measures may be adopted to mitigate possible misuses of smart things capabilities. However, due to different reasons, such as conflicting business incentives, undemocratic governments, or users'

ignorance these legal measures may not be enforceable. The alternative is to preclude possible misuses by design. Studies of core Internet protocols show that protocol characteristics have significant political, economic, and social implications [85]. UMA protocol may also have significant implications as it challenges currently well established business practices. However, as UMA is a novel, not yet widely adopted solution, it is too early to estimate extent of these implications. Our solution exploits UMA protocol features to obviate the need for fully trusting various service providers. Consequently, it empowers users to control data dissemination as ultimately they are the most appropriate entity to make decision whether their data can be accessed and how it will be accessed.

Bibliography

[1] "The social web of things," Mar. 2011. [Online]. Available: http://www.youtube.com/watch?v=i5AuzQXBsG4

[2] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas, "Towards security and privacy for pervasive computing," in *Proceedings of the 2002 Mext-NSF-JSPS international conference on Software security: theories and systems*, ser. ISSS'02. Berlin, Heidelberg: Springer-Verlag, 2003. ISBN 3-540-00708-3 pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1765533.1765535

[3] P. Ahonen, P. Alahuhta, B. Daskala, S. Delaitre, P. D. Hert, R. Lindner, I. Maghiros, A. Moscibroda, W. Schreurs, and M. Verlinden, *Safeguards in a World of Ambient Intelligence*, ser. The International Library of Ethics, Law and Technology, D. Wright, M. Friedewald, Y. Punie, S. Gutwirth, and E. Vildjiounaite, Eds. Springer, 2010, vol. 1, DOI: 10.1007/978-1-4020-6662-7. [Online]. Available: http://www.springerlink.com/content/978-90-481-8786-7/#section=145633&page=1

[4] R. H. Weber, "Internet of things - new security and privacy challenges," *Computer Law & Security Review*, vol. 26, no. 1, pp. 23–30, 2010. doi: 10.1016/j.clsr.2009.11.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364909001939

[5] Chester Wisniewski, "Smart meter hacking can disclose which TV shows and movies you watch," Jan. 2012. [Online]. Available: http://nakedsecurity.sophos.com/2012/01/08/28c3-smart-meter-hacking-can-disclose-which-tv-shows-and-movies-you-watch/

[6] Dan Goodin, "How an internet-connected samsung TV can spill your deepest secrets," Dec. 2012.

[Online]. Available: http://arstechnica.com/security/2012/12/how-an-internet-connected-samsung-tv-can-spill-your-deepest-secrets/

[7] Lesley Ciarula Taylor, "U.S. chamber of commerce details 'sophisticated' 6-month hacking of its computers," *The Toronto Star*, Dec. 2011. [Online]. Available: http://www.thestar.com/news/world/2011/12/21/us_chamber_of_commerce_details_sophisticated_6month_hacking_of_its_computers.html

[8] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, Jul. 2012. doi: 10.1109/IMIS.2012.104 pp. 751–756.

[9] B. Ostermaier, F. Schlup, and K. Romer, "WebPlug: a framework for the web of things," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010. doi: 10.1109/PERCOMW.2010.5470522 pp. 690–695.

[10] Ericsson, "More than 50 billion connected devices," *White Paper*, Feb. 2011. [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf

[11] Dave Evans, "The internet of things: How the next evolution of the internet is changing everything," Apr. 2011. [Online]. Available: http://www.cisco.com/web/about/ac79/iot/index.html

[12] T. Bauge, C. Sorge, A. Waller, G. Selander, J. Bohli, O. Ugus, and D. Williams, *Security and Accounting for SENSEI*, Jun. 2010. [Online]. Available: http://www.ict-sensei.org/index.php?option=com_chronocontact&chronoformname=SENSEI_WP3_D3.5

[13] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffle, "Vision and challenges for realising the internet of things," *CERP-IoT, European Commission, Luxembourg*, 2010. [Online]. Available: http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf

[14] Luigi Atzori, Antonio Iera, and Giacomo Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. doi: 10.1016/j.comnet.2010.05.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[15] E. Fleisch, "What is the internet of things? an economic perspective," *Auto-ID Labs Whitepaper*, Jan. 2010. [Online]. Available: http://www.wew.autoidlabs.org/uploads/media/AUTOIDLABS-WP-BIZAPP-53.pdf

[16] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009.

[17] Dominique Guinard, "A web of things application architecture - integrating the real-world into the web," Ph.D., ETH Zurich, 2011. [Online]. Available: http://webofthings.org/dom/thesis.pdf

[18] Vlad Trifa, "Building blocks for a participatory web of things: Devices, infrastructures, and programming frameworks," Ph.D., ETH Zurich, 2011. [Online]. Available: http://vladtrifa.com/research/files/phd-thesis.pdf

[19] "ThingWorx: M2M application platform," Mar. 2013. [Online]. Available: http://www.thingworx.com/

[20] "Axeda: Machine cloud & M2M platform." [Online]. Available: http://www.axeda.com/

[21] "Lockitron: Internet connected door lock." [Online]. Available: https://lockitron.com/

[22] "LIFX: the light bulb reinvented." [Online]. Available: http://www.kickstarter.com/projects/limemouse/lifx-the-light-bulb-reinvented

[23] F. Carrez, M. Bauer, T. Bauge, and J. Bernat, *SENSEI Reference Architecture*, Jan. 2008. [Online]. Available: http://www.ict-sensei.org/index.php?option=com_chronocontact&chronoformname=SENSEI_WP3_D3.2

[24] "Resource exhaustion attack." [Online]. Available: https://www.owasp.org/index.php/Resource_exhaustion

[25] E. Maler and T. Hardjono, "Binding obligations on user-managed access (UMA) participants," *Internet-Draft*, vol. draft-maler-uma-trust-00, Jan. 2013. [Online]. Available: http://docs.kantarainitiative.org/uma/draft-uma-trust.html

[26] R. H. Weber, "Internet of things - need for a new legal environment?" *Computer Law & Security Review*, vol. 25, no. 6, pp. 522–527, Nov. 2009. doi: 10.1016/j.clsr.2009.09.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364909001514

[27] R. Weber, "Accountability in the internet of things," *Computer Law & Security Review*, vol. 27, no. 2, pp. 133–138, Apr. 2011. doi: 10.1016/j.clsr.2011.01.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0267364911000069

[28] M. Presser, P. Barnaghi, M. Eurich, and C. Villalonga, "The SENSEI project: integrating the physical world with the digital world of the network of the future," *IEEE Communications Magazine*, vol. 47, no. 4, pp. 1–4, Apr. 2009. doi: 10.1109/MCOM.2009.4907403

[29] Craig Burton, Scott David, Drummond Reed, Doc Searls, and Phillip J. Windley, "From personal computers to personal clouds: The advent of the cloud OS," Apr. 2012. [Online]. Available: http://www.windley.com/docs/2012/cloudos.pdf

[30] P. Viswanathan, B. Gill, and R. Campbell, "Security architecture in gaia," University of Illinois at Urbana-Champaign, Champaign, IL, USA, Tech. Rep., 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=870854

[31] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and M. Mickunas, "Cerberus: a context-aware security scheme for smart spaces," in *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, 2003, pp. 489–496.

[32] M. Covington, P. Fogla, Z. Zhan, and M. Ahamad, "A context-aware security architecture for emerging applications," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, 2002, pp. 249–258.

[33] D. Guinard, M. Fischer, and V. Trifa, "Sharing using social networks in a composable web of things," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE, Mar. 2010. doi: 10.1109/PERCOMW.2010.5470524. ISBN 978-1-4244-6605-4 pp. 702–707.

[34] Kim, J.E., Boulos, G., Yackovich, J., Barth, T., and Mosse, D., "Seamless integration of heterogeneous devices and access control in

smart homes," in *Proceedings of the 8th International Conference on Intelligent Environments (IE 2012). Guanajuato, Mexico*, 2012. [Online]. Available: http://vs.inf.ethz.ch/publ/papers/beckel-2012-seamless.pdf

[35] "Xively." [Online]. Available: https://xively.com

[36] "Paraimpu: The web of things is more than things in the web." [Online]. Available: http://paraimpu.crs4.it/

[37] A. Pintus, D. Carboni, and A. Piras, "The anatomy of a large scale social web for internet enabled objects," in *Proceedings of the Second International Workshop on Web of Things*, ser. WoT '11. New York, NY, USA: ACM, 2011. doi: 10.1145/1993966.1993975. ISBN 978-1-4503-0624-9 pp. 6:1–6:6. [Online]. Available: http://doi.acm.org/10.1145/1993966.1993975

[38] "ThingSpeak: internet of things application platform." [Online]. Available: https://www.thingspeak.com/

[39] "Xively: Device management API." [Online]. Available: https://xively.com/dev/docs/api/product_management/provisioning/

[40] M. Benantar, *Access Control Systems: Security, Identity Management and Trust Models*. Springer, Jun. 2006. ISBN 9780387277165

[41] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, "RFC 2904: AAA authorization framework," *Network Working Group, The Internet Society*, 2000. [Online]. Available: http://tools.ietf.org/html/rfc2904

[42] B. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

[43] H. Stiegler, "A structure for access control lists," *Software: Practice and Experience*, vol. 9, no. 10, pp. 813–819, 1979.

[44] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[45] A. INCITS, "INCITS 359-2004," *Role based access control*, 2004.

[46] Tim Moses, Ed., *eXtensible Access Control Markup Language (XACML) Version 2.0*, committee specification ed. OASIS, Feb. 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[47] Q. Pham, J. Reid, A. McCullagh, and E. Dawson, "On a taxonomy of delegation," *Computers & Security*, vol. 29, no. 5, pp. 565–579, Jul. 2010. doi: 10.1016/j.cose.2009.12.009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404809001473

[48] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (v5)," *RFC 4120 (Standards Track)*, Jul. 2005. [Online]. Available: http://tools.ietf.org/html/rfc4120

[49] B. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33 –38, Sep. 1994. doi: 10.1109/35.312841

[50] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and protocols for the OASIS security assertion markup language (SAML) v2.0," *OASIS*, vol. Standard, Mar. 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[51] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo, "Security assertion markup language (SAML) v2.0 technical overview," *OASIS*, vol. Committee Draft 02, Mar. 2008. [Online]. Available: http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[52] Frederick Hirsch, Rob Philpott, and Eve Maler, "Security and privacy considerations for the OASIS security assertion markup language (SAML) v2.0," *OASIS*, vol. Standard, Mar. 2005.

[53] Sun Microsystems, Inc., "SunXACML," Jul. 2004. [Online]. Available: http://sunxacml.sourceforge.net/

[54] "UMU XACML editor." [Online]. Available: http://xacml.dif.um.es/

[55] "Xacml studio: An authorization policy editor." [Online]. Available: http://xacml-studio.sourceforge.net/

[56] E. Hammer-Lahav, "The OAuth 1.0 protocol," *RFC 5849 (Informational)*, Apr. 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5849.txt

[57] D. Hardt and D. Recordon, "The OAuth 2.0 authorization framework," *RFC 6749*, Oct. 2012. [Online]. Available: http://tools.ietf.org/html/rfc6749

[58] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web APIs on the world wide web," in *2010 IEEE 8th European Conference on Web Services (ECOWS)*, Dec. 2010. doi: 10.1109/ECOWS.2010.9 pp. 107 – 114.

[59] ProgrammableWeb.com, "API dashboard: ProgrammableWeb," Mar. 2013. [Online]. Available: http://www.programmableweb.com/apis

[60] T. Hardjono, "User-managed access (UMA) profile of OAuth 2.0," *Internet-Draft*, vol. draft-hardjono-oauth-umacore-07b, Jan. 2013. [Online]. Available: http://docs.kantarainitiative.org/uma/draft-uma-core.html

[61] M. P. Machulak, L. Moren, and A. van Moorsel, "Design and implementation of user-managed access framework for web 2.0 applications," in *Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing*, ser. MW4SOC '10. New York, NY, USA: ACM, 2010. doi: 10.1145/1890912.1890913. ISBN 978-1-4503-0452-8 pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1890912.1890913

[62] M. P. Machulak, E. L. Maler, Domenico Catalano, and Aad van Moorsel, "User-managed access to web resources," in *Proceedings of the 6th ACM workshop on Digital identity management*, ser. DIM '10. New York, NY, USA: ACM, 2010. doi: 10.1145/1866855.1866865. ISBN 978-1-4503-0090-2 pp. 35–44. [Online]. Available: http://doi.acm.org/10.1145/1866855.1866865

[63] "User managed access work group." [Online]. Available: http://kantarainitiative.org/groups/user-managed-access-work-group/

[64] "Kantara initiative." [Online]. Available: http://kantarainitiative.org

[65] T. Hardjono, "OAuth 2.0 resource set registration," *Internet-Draft*, vol. draft-hardjono-oauth-resource-reg-00, Dec. 2013. [Online]. Available: http://tools.ietf.org/html/draft-hardjono-oauth-resource-reg-00

[66] T. Hardjono, M. Machulak, E. Maler, and C. Scholz, "OAuth dynamic client registration protocol," *Internet-Draft*, vol. draft-oauth-dyn-reg-v1-03 (work in progress), Oct. 2011. [Online]. Available: http://tools.ietf.org/html/draft-oauth-dyn-reg-v1-03

[67] N. Sakimura, J. Bradley, and M. Jones, "OpenID connect dynamic client registration 1.0," May 2012. [Online]. Available: http://openid.net/specs/openid-connect-registration-1_0.html

[68] B. Sarikaya, Y. Ohba, R. Moskovitz, Z. Cao, and R. Cragie, "Security bootstrapping solution for resource-constrained devices," *Internet-Draft*, vol. draft-sarikaya-core-sbootstrapping-05 (work in progress), Jul. 2012. [Online]. Available: http://tools.ietf.org/html/draft-sarikaya-core-sbootstrapping-05

[69] "UMA implementations." [Online]. Available: http://kantarainitiative.org/confluence/display/uma/UMA+Implementations

[70] "Puma," Apr. 2012. [Online]. Available: http://smartjisc.wordpress.com/2012/04/13/releasing-puma/

[71] P. Mularien, *Spring Security 3*. Packt Publishing Ltd, 2010. ISBN 9781847199751

[72] "Spring framework." [Online]. Available: http://www.springsource.org/spring-framework

[73] "Spring security." [Online]. Available: http://www.springsource.org/spring-security

[74] "Spring security OAuth." [Online]. Available: http://www.springsource.org/spring-security-oauth

[75] "Cloud foundry." [Online]. Available: http://www.cloudfoundry.com/

[76] "MyBatis." [Online]. Available: http://mybatis.github.com/mybatis-3/

[77] T. Lodderstedt, M. McGloin, and P. Hunt, "OAuth 2.0 threat model and security considerations," *RFC 6819 (Informational)*, Jan. 2013. [Online]. Available: http://tools.ietf.org/html/rfc6819

[78] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," *RFC 5246 (Standards Track)*, Aug. 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246

[79] E. Rescorla, "HTTP over TLS," *RFC 2818 (Informational)*, May 2000. [Online]. Available: http://tools.ietf.org/html/rfc2818

[80] G. Denaro, A. Polini, and W. Emmerich, "Early performance testing of distributed software applications," in *Proceedings of the 4th international workshop on Software and performance*, ser. WOSP '04. New York, NY, USA: ACM, 2004. doi: 10.1145/974044.974059. ISBN 1-58113-673-0 pp. 94–103. [Online]. Available: http://doi.acm.org/10.1145/974044.974059

[81] O. Garcia-Morchon, S. Keoh, S. Kumar, R. Hummen, and R. Struik, "Security considerations in the IP-based internet of things," *Internet-Draft*, vol. draft-garcia-core-security-04 (work in progress), Mar. 2012. [Online]. Available: http://tools.ietf.org/html/draft-garcia-core-security-04

[82] J. Lanier, *Who Owns the Future?* New York, NY, USA: Simon & Schuster, May 2013. ISBN 9781451654967

[83] World Economic Forum, "Personal data: The emergence of a new asset class," Jan. 2011. [Online]. Available: http://www.weforum.org/reports/personal-data-emergence-new-asset-class

[84] Martin Kuppinger, "Life management platforms: Control and privacy for personal data," Mar. 2012. [Online]. Available: http://www.kuppingercole.com/report/advisorylifemanagementplatforms7060813412

[85] A. R. Galloway, *Protocol: How Control Exists After Decentralization*. MIT Press, 2004. ISBN 9780262072472

TRITA-ICT-EX-2013:231

www.kth.se